



UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Investigating Optimisations of Dynamic Time Warping for Keyword Spotting

Geoffrey T. Frost
20806256

Report submitted in partial fulfilment of the requirements of the module
Project (E) 448 for the degree Baccalaureus in Engineering in the Department of
Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Prof T.R. Niesler

November 2020

Acknowledgements

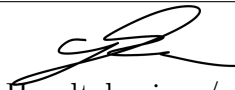
I would like to express my gratitude to my supervisor Prof TR Niesler who's feedback was instrumental. Members of the DSP lab who unveiled themselves to answer questions regarding the current DTW keyword spotter, as well as supplying the relevant data to conduct our experiments. My parents, whom without their support I would not have had this opportunity. And lastly, my friends and fellow members of the "Yeet Fleet" engineering group, who spent many hours on Discord offering their unwavering support throughout online classes and Covid-19.



UNIVERSITEIT • STELLENBOSCH • UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
Plagiarism is the use of ideas, material and other intellectual property of another's work and to present it as my own.
2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
I agree that plagiarism is a punishable offence because it constitutes theft.
3. Ek verstaan ook dat direkte vertalings plagiaat is.
I also understand that direct translations are plagiarism.
4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelike aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

20806256 Studentenommer / <i>Student number</i>	 Handtekening / <i>Signature</i>
G.T. Frost Voorletters en van / <i>Initials and surname</i>	08/11/2020 Datum / <i>Date</i>

Abstract

English

Traditional ASR techniques struggle when applied to low-resource languages where the lack of data severely impacts their performance. To circumvent the lack of data for such low-resource languages, a current research project is utilising DTW as means to measure the similarity between labelled keywords and segments of spoken utterances, a process described as keyword spotting. However, a major bottleneck in the system is the computational complexity of DTW. For this reason, this project investigated pre-existing DTW optimisations, and formulated new optimisations that utilised the keyword spotter architecture. These were subsequently compared to a baseline system with the use of relevant metrics. The results obtained from these experiments are promising and show that with little to no impact on performance, a runtime speed-up can be achieved. Notably, one of the formulated optimisations achieved a speed-up factor larger than an order of magnitude, whilst only a 1% degradation in AUC was observed, and a 1.8% improvement in EER.

Afrikaans

Tradisionele ASR-tegnieke sukkel wanneer dit toegepas word op lae-hulpbron tale, waar die gebrek aan data hul prestasie ernstig beïnvloed. Om die gebrek aan data vir sulke lae-hulpbron tale te omseil, gebruik DTW 'n huidige navorsingsprojek as middel om die ooreenkoms tussen benoemde sleutelwoorde en segmente van gesproke uitings te meet, 'n proses wat beskryf word as sleutelwoord bespeuring. 'n Groot knelpunt in die stelsel is egter die berekening van die kompleksiteit van DTW. Om hierdie rede is voorafbestaande DTW optimerings ondersoek en nuwe optimerings geformuleer wat gebruik maak van die sleutelwoord argitektuur. Dit is vervolgens vergelyk met 'n basislynstelsel deur gebruik te maak van relevante statistieke. Die resultate verkry uit hierdie eksperimente is belowend en toon dat 'n merkbare verbetering in looptyd met min of geen impak op die prestasie behaal kan word. Opmerklik is dat een van die geformuleerde optimerings 'n versnellingsfaktor van meer as 'n ordegrootte getoon het, terwyl die waargenome AUC met net 1% versleg het, en die EER met 1.8% verbeter het.

Contents

Declaration	ii
Abstract	iii
List of Figures	vii
List of Tables	ix
Nomenclature	x
1. Introduction	1
1.1. Project Background	1
1.2. Motivation	1
1.3. Objectives	2
2. Literature Study	3
2.1. Dynamic time warping	3
2.1.1. Algorithm	3
2.1.2. Applications	5
2.2. Existing optimisation methods	6
2.2.1. Sakoe-Chiba band	7
2.2.2. Itakura parallelogram	8
2.2.3. FastDTW	10
2.2.4. Parallelisation	11
2.3. DTW Keyword Spotter	12

2.4. Acoustic Feature Vectors	13
2.4.1. Mel frequency cepstral coefficients	14
2.5. Chapter summary	15
3. Proposed optimisations for keyword spotting	16
3.1. Cached D DTW	16
3.2. P^* DTW	17
3.3. Grad DTW	18
3.4. GRAM DTW	19
3.5. Chapter summary	20
4. Experimental setup	21
4.1. Evaluation metrics	21
4.1.1. Receiver operating characteristic	22
4.1.2. Area under the curve	22
4.1.3. Equal error rate	22
4.1.4. Runtime and speed-up factor	23
4.2. Data	23
4.2.1. Keywords templates	23
4.2.2. SABN corpus	24
4.2.3. MFCCs	24
4.3. Architecture	24
4.3.1. Just-In-Time compiler	25
4.4. Keyword spotter	25
4.5. Baseline	26
4.6. Implemented optimisations	26
4.6.1. S-C band	27
4.6.2. Itakura band	28

4.6.3. P^* DTW	29
4.6.4. Cached D DTW	29
4.6.5. Grad DTW	30
4.6.6. GRAM DTW	31
4.6.7. Summary	32
4.7. Chapter summary	32
5. Experimental results and discussion	33
5.1. Results	33
5.1.1. S-C band	33
5.1.2. Itakura band	34
5.1.3. P^* DTW	34
5.1.4. Cached D DTW	34
5.1.5. Grad DTW	35
5.1.6. GRAM DTW	35
5.2. Summary of results	35
5.3. Discussion	36
5.3.1. Observations	37
5.3.2. Performance	37
5.4. Chapter summary	38
6. Summary and Conclusion	39
6.1. Future work	40
Bibliography	41
A. Project Planning Schedule	43
B. Outcomes Compliance	44

List of Figures

2.1.	An illustration of the application of DTW to two scalar time series \mathbf{x} and \mathbf{y} .	4
2.2.	Global constraint regions for two optimisations: S-C band and the Itakura parallelogram.	7
2.3.	Approximate theoretical speed-up factors for both constraint region optimisations as their respective hyperparameters are increased.	8
2.4.	The four different resolutions evaluated during a complete run of the Fast-DTW algorithm on two temporal sequences of length 16. Dark line indicates the current warping path, dark grey cells show the expanded P'^* and the light grey cells show the additional cells considered as per the <i>radius</i> parameters. [1]	11
2.5.	An illustration of how to populate γ along the diagonals. Arrows show dependencies. The diagonals shaded from dark to light are denoted as d_0 , d_1 , and d_2 respectively. Elements along d_0 can be computed in parallel. . .	12
2.6.	An illustration of the DTW keyword spotting architecture. For each window of overlap the DTW is computed and a similarity score J is produced, this process is then repeated for each repetition for each keyword. Once this is complete the largest similarity score is determined and if it is larger than a predetermined threshold, it is decided that the utterance contains the corresponding keyword.	13
2.7.	Spectrum of Mel filter banks and the corresponding Mel frequency mapping [2].	14
3.1.	A high level diagram illustrating the key concepts of Cached D DTW. . . .	16
3.2.	Illustrations of the global constraint regions imposed by P^* DTW for temporal sequences of length N	17
3.3.	An example illustrating the DTW costs (c) as a keyword is approached, and how the DTW cost fluctuates when there is no keyword present in the window of overlap.	19
4.1.	A confusion matrix depicting the ground truth and predicted labels for TP, TN, FP, and FN decisions. Cells highlighted in green indicate metrics where higher values indicate a better classifier, whereas red indicates metrics where lower values infer better performance.	22

4.2.	Representation of FPR and FNR as the decision threshold is increased. The error rate where FPR and FNR cross is labelled the EER.	23
4.3.	ROC plots for both the existing DTW keyword spotter and our implementation.	26
4.4.	AUC and EER on the development set for increasing values of r for the S-C band DTW optimisation.	27
4.5.	AUC and EER on the development set for increasing values of m_1 for the Itakura band DTW optimisation.	28
4.6.	AUC and EER on the development set for increasing values of r for the P^* DTW optimisation.	29
4.7.	AUC and EER on the development set for increasing values of ζ_{\max} for the Grad DTW optimisation.	30
4.8.	AUC and EER for increasing values of ζ_{\max} for the GRAM DTW optimisation.	31
5.1.	A composition of ROC plots for the results generated by Classic DTW and all the implemented optimisations.	36

List of Tables

4.1.	The SABN subsets generated for development and testing, indicating the the number of utterances that contain a keyword (kw) and those that do not. The total duration is also given in hours (h).	24
4.2.	Baseline results for classic DTW on the development set. Values are expressed as percentages (%), i.e. an AUC of 100% represents an area under the curve of 1.	26
4.3.	Summary of the final values of all hyperparameters for each optimisation. .	32
5.1.	Development and test set results for the S-C band optimisation. Runtime and speed-up are with reference to the test set only.	33
5.2.	Development and test set results for the Itakura band optimisation. Runtime and speed-up are with reference to the test set only.	34
5.3.	Development and test set results for the P^* DTW optimisation. Runtime and speed-up are with reference to the test set only.	34
5.4.	Development and test set results for cached D DTW. Runtime and speed-up are with reference to the test set only.	34
5.5.	Development and test set results for Grad DTW. Runtime and speed-up are with reference to the test set only.	35
5.6.	Development and test set results for GRAM DTW. Runtime and speed-up are with reference to the test set only.	35
5.7.	Development and test set results. The best scores for each metric are shown in bold. Runtime and speed-up are with reference to the test set only. . . .	36
A.1.	Project goals and planned time frame.	43
B.1.	ECSA outcomes and how they are achieved.	44
B.2.	ECSA outcomes and how they are achieved. (continued)	45

Nomenclature

Variables and functions

D	Pairwise vector distance matrix.
d	An arbitrary distance function.
γ	Minimum cumulative distance matrix.
P^*	Optimal warping path.
R	Global constraint region.
r	Bandwidth for the Sakoe-Chiba band, and the radius of the P^* optimisation.
P'^*	Optimal warping path from the previous lower resolution.
ζ	Frame skip.
Δ_c	Approximate gradient of the last two DTW costs.
β	The momentum term in GRAM DTW.
k	Scaling factor for the GRAM DTW momentum equation.

Acronyms and abbreviations

DTW	Dynamic time warping
ASR	Automatic speech recognition
GPU	Graphics processing unit
CNN	Convolutions neural networks
UN	United Nations
CPU	Central processing unit
KWS	Keyword spotting
MFCCs	Mel frequency cepstral coefficients

Chapter 1

Introduction

1.1. Project Background

Automatic speech recognition (ASR) has been a focus of study for researchers for decades. It is a method that hinges on converting speech audio to output lattices that represent hypothesised transcriptions and text, which can subsequently be searched for desired information. Recent improvements in parallel processing hardware such as graphics processing units (GPU)s has allowed the use of complex machine learning algorithms to be utilised to solve ASR problems. One drawback of these techniques is that to have a well-performing system, a large amount of training data is required. This is not an issue for languages such as English where there is an abundance of transcribed speech. However, for under-resourced languages (such as Acholi, a language spoken by remote communities in Uganda), the lack of data means these state of the art algorithms cannot be utilised.

Dynamic time warping (DTW) proved to be a popular method in ASR systems a few decades ago. DTW is a method used to optimally align two temporal sequences, accomplished by a non-linear mapping between their respective time axes. Due to the non-linear alignment, DTW can capture the similarity between two sequences despite the presence of distortions or phase offset. In addition to alignment, DTW also computes a distance score between the sequences. In the case of speech signals, this can be used as a metric to quantify the similarity between two utterances.

Due to its $O(N^2)$ time complexity and inability to model statistical variability, its use in the field receded as more sophisticated algorithms emerged. However, DTW has recently demonstrated renewed usefulness in low-resource ASR-free keyword spotting applications, where conventional ASR techniques such as convolutional neural networks (CNNs) struggle due to the limited labelled training data synonymous with a low-resource language.

1.2. Motivation

Currently, the Digital Signal Processing (DSP) lab at Stellenbosch University is developing a keyword spotter for the United Nations (UN). In communities where there is little access to social media, local radio phone-in broadcast often serve as a vital platform for members of these communities to voice issues of pressing importance. For the UN to

better understand where humanitarian aid is needed, they need to be made aware of these issues, which is greatly supported the automatic monitoring of these broadcasts for specific keywords. Often in the case of remote communities, there are little to no resources in the local language, which means traditional ASR techniques cannot be applied. In such cases, DTW can be used to compute the similarity between chosen keywords and the contents of a target utterance. To accomplish this, only a limited number of labelled keywords are required and most importantly, the system requires no training and hence no transcribed data. However, DTW is computational expensive which makes it difficult to implement such a system practically. For this reason, it is necessary to investigate optimisations of DTW in a keyword spotter system to reduce overall runtime.

1.3. Objectives

This project will investigate the effectiveness of existing DTW optimisations for the application of DTW keyword spotting. It will propose and evaluate several new DTW optimisations that take advantage of the keyword spotting architecture. All optimisations will be compared to a baseline system, where their performance will be evaluated with relevant metrics.

Chapter 2

Literature Study

In this chapter we will discuss the fundamental concepts of DTW, and provide brief summaries of some recent interesting applications. We also discuss and analyse some existing optimisations. Since the goal of this project is to investigate optimisations of DTW for keyword spotting, we also present an in-depth description of the DTW keyword spotter. Lastly, we highlight the importance of acoustic feature vectors, and describe our chosen ones.

2.1. Dynamic time warping

DTW was first developed in 1978 by Sakoe and Chiba for spoken word recognition [3]. DTW's non-linear mapping allowed for the comparison between utterances and the pre-recorded keyword templates despite timing and pronunciation variations, yielding accurate predictions.

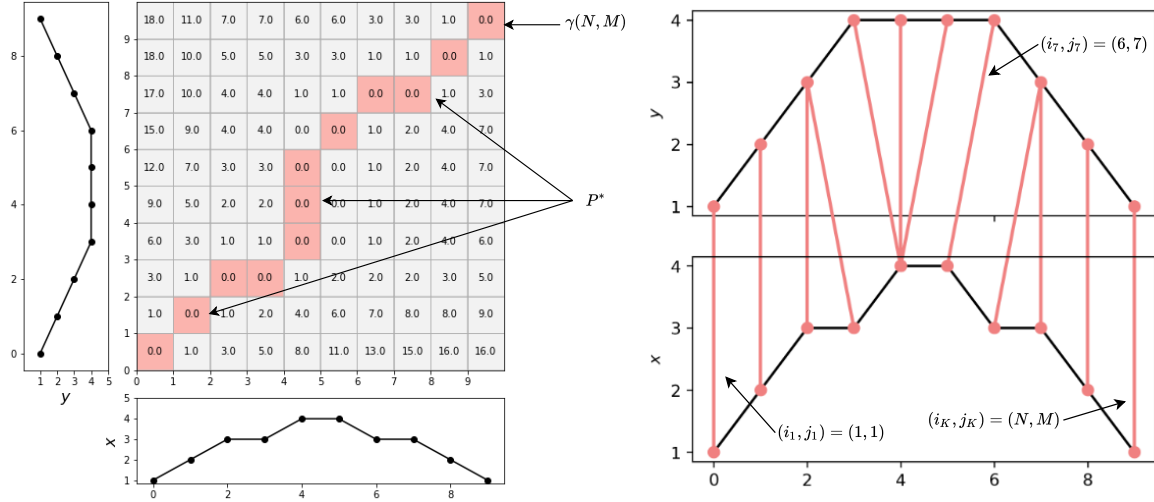
2.1.1. Algorithm

DTW aims to align two temporal sequences, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_M]$ where both \mathbf{x}_i and $\mathbf{y}_j \in \mathbb{R}^d$. A distance matrix $D \in \mathbb{R}^{N \times M}$ is populated by computing the pairwise vector distances $D(i, j) = d(\mathbf{x}_i, \mathbf{y}_j)$, where d is an arbitrary distance function [4]. The minimum cumulative distance matrix $\gamma \in \mathbb{R}^{N \times M}$, which contains the cumulative distance for each point, is calculated according to the following recurrence relation [5], which is a dynamic programming approach:

$$\gamma(i, j) = D(i, j) + \min[\gamma(i-1, j), \gamma(i, j-1), \gamma(i-1, j-1)] \quad (2.1)$$

With the following boundary conditions:

1. $\gamma(1, 1) = D(1, 1)$
2. $\gamma(1, j) = \gamma(1, j-1) + D(1, j)$
3. $\gamma(i, 1) = \gamma(i-1, 1) + D(i, 1)$



(a) An example of the matrix γ for two time series \mathbf{x} and \mathbf{y} and the corresponding optimal warping path. (b) An illustration of P^* being used to align two time series \mathbf{x} and \mathbf{y} .

Figure 2.1: An illustration of the application of DTW to two scalar time series \mathbf{x} and \mathbf{y} .

Upon completion of this recursive computation of the matrix γ , the optimal warping path P^* , which is represented as a series of index pairs $[(i_1, j_1), \dots, (i_K, j_K)]$ for $k = 1 \dots K$, can be determined by tracing backwards from $\gamma(N, M)$ [5] [6]. This optimal warping path is the best (lowest cost) alignment between \mathbf{X} and \mathbf{Y} . The traceback is accomplished by starting at $\gamma(N, M)$ and repeatedly choosing the previous point with the smallest cumulative distance while adhering to the following constraints [5]:

1. $(i_1, j_1) = (1, 1)$
2. $(i_K, j_K) = (N, M)$
3. $0 \leq i_{k+1} - i_k \leq 1$ and $0 \leq j_{k+1} - j_k \leq 1$ for all $k < K$

Lastly, the DTW score between the two temporal sequences is denoted as c shown in Equation 2.2:

$$c = \gamma(N, M) = \text{DTW}(\mathbf{X}, \mathbf{Y}) \quad (2.2)$$

The concept of the matrix γ and P^* are illustrated in Figure 2.1a for two example scalar time series, \mathbf{x} and \mathbf{y} . The alignment of these two time series is presented in Figure 2.1b. Note that the aforementioned constraints are observed. Pseudo code for computing γ and P^* can be found below. Algorithm 2.1 computes the matrix γ as described by Equation 2.1, while Algorithm 2.2 performs the backtracking previously described.

Algorithm 2.1: DTW accumulative cost matrix algorithm

```

 $D \leftarrow d(\mathbf{X}, \mathbf{Y})$ 
 $\gamma(1, 1) \leftarrow D(1, 1)$ 
for  $i = 1, 2, \dots, N$  do
     $\gamma(i, 1) \leftarrow D(i, 1) + \gamma(i - 1, 1)$ 
end for
for  $j = 1, 2, \dots, M$  do
     $\gamma(1, j) \leftarrow D(1, j) + \gamma(1, j - 1)$ 
end for
for  $i = 1, 2, \dots, N$  do
    for  $j = 1, 2, \dots, M$  do
         $\gamma(i, j) \leftarrow D(i, j) + \min[\gamma(1, j - 1), \gamma(i - 1, j), \gamma(i - 1, j - 1)]$ 
    end for
end for

```

Algorithm 2.2: DTW optimal warping path algorithm

```

 $i \leftarrow N$ 
 $j \leftarrow M$ 
 $steps \leftarrow [(1, 0), (0, 1), (1, 1)]$ 
 $P^*.append(i, j)$ 
while  $i > 1$  and  $j > 1$  do
    if  $i > 1$  and  $j > 1$  then
         $(i, j) \leftarrow (i, j) - steps[\text{argmin}([\gamma(i - 1, j), \gamma(i, j - 1), \gamma(i - 1, j - 1)])]$ 
    end if
    if  $i > 1$  and  $j = 1$  then
         $(i, j) \leftarrow (i, j) - (1, 0)$ 
    end if
    if  $i = 1$  and  $j > 1$  then
         $(i, j) \leftarrow (i, j) - (0, 1)$ 
    end if
     $P^*.append(i, j)$ 
end while

```

Due to the nested **for** loops in Algorithm 2.1, DTW has a runtime complexity of $O(NM)$ or more simply $O(N^2)$ assuming $N \approx M$.

2.1.2. Applications

Since its proposal by Sakoe and Chiba in 1978, the DTW algorithm has been applied to many problems. Some recent applications that utilise DTW are outlined below.

Early detection method for Alzheimer’s disease

DTW has been used in an early detection method for Alzheimer disease [7]. This is accomplished by recording the foot movements of an individual, and then computing the DTW between these recordings and a set of reference measurements from individuals without Alzheimer’s disease. If the resulting DTW distance is higher than a predetermined threshold, Alzheimer’s disease is suspected.

Human gesture recognition

In a more genral application, DTW has been utilised in simple human gesture recognition [8]. A feature vector is extracted from an array of wearable sensors, after which the temporal sequence is compared to a comprehensive list of templates, the pair with the smallest DTW distance is used to label the gesture, provided it is lower than a predetermined threshold.

Musical recording alignment

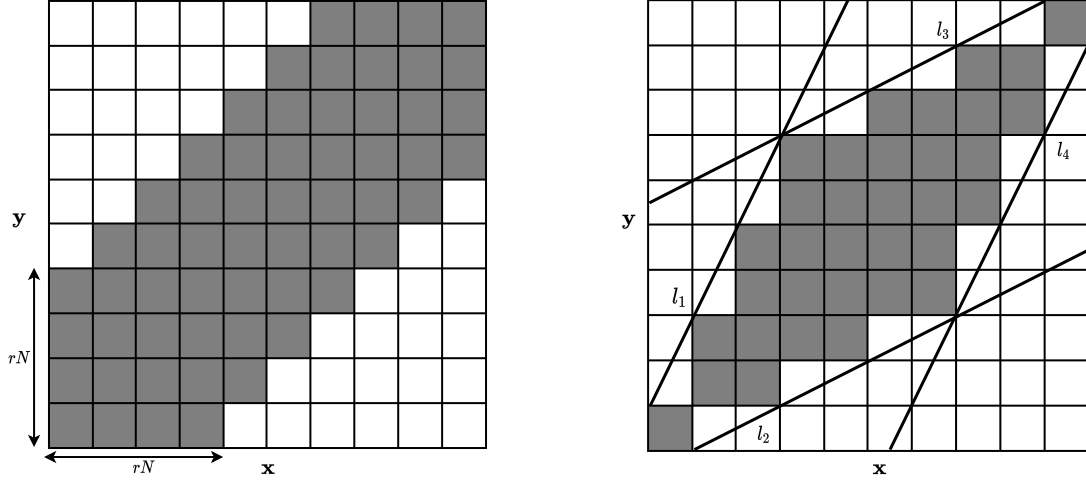
Whilst the previous examples have highlighted DTW’s applications in classification problems, it is also a popular tool in temporal sequence alignment. In music research, there is a need to align different recordings of the same musical piece [9]. Due to DTW’s ability to perform non-linear alignment, it can align two recordings that might have slight structural differences, a common occurrence resulting from live performances. In another musically related application, DTW is used to uncover emotionally related musical works (works that make listeners feel similar emotions) [10].

Humanitarian keyword spotting

Recently, and from which the focus of this study stems, DTW has been utilised in a keyword spotting application for the United Nations (UN) [11]. Whilst traditional keyword spotting techniques require large amounts of data to train complex models, DTW only requires a few pre-recorded keywords, which can then be used as templates when analysing utterances in a process which is detailed in Section 2.3. This is to be applied in remote regions of Africa, where almost no language resources are available, and hence the required data to train complex models does not exist. However, due to DTW’s $O(N^2)$ time complexity it is difficult to use the algorithm in real-time, and hence the focus of this study is to investigate optimisations specific to this problem.

2.2. Existing optimisation methods

Due to DTW’s $O(N^2)$ time complexity, its use in keyword spotting is limited. For this reason, it is necessary to investigate optimisation techniques to reduce this time complexity.



(a) Illustration of the global constraint region R (grey shaded region) imposed by the S-C band on γ . r indicates the band width.

(b) Illustration of the global constraint region R (grey shaded region) imposed by the Itakura parallelogram on γ . l_1 , l_2 , l_3 , and l_4 indicate the four lines used to construct the constraint region.

Figure 2.2: Global constraint regions for two optimisations: S-C band and the Itakura parallelogram.

In this section, we detail some popular existing optimisation methods used for DTW.

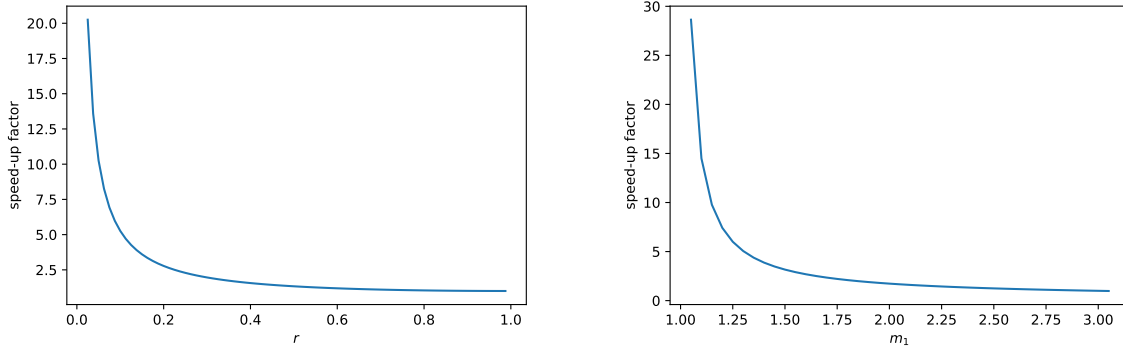
2.2.1. Sakoe-Chiba band

The Sakoe-Chiba band (S-C band) was first introduced as a means to stop DTW from finding unlikely warping paths [3]. However, it has the added benefit of reducing the total number of computations by introducing a simple constraint: Only compute γ for the region $R = \{i, j\}$ such that, $j - r \leq i \leq j + r$, where $1 \leq i \leq N$ and $1 \leq j \leq M$ and r is the band width as illustrated in Figure 2.2a. In this application r is expressed as fraction of the length of the temporal sequences such that $r \in [0, 1]$. Thus the width of the band illustrated in Figure 2.2a becomes rN .

We now derive expressions for the reduction in computation achieved by using the S-C band. With reference to Figure 2.2a, The total of the matrix γ 's elements not computed when using this optimisation can be determined by calculating the number of elements in the upper and lower triangles excluded by the constraint region. The area of each of these triangles is $\frac{1}{2}(N - rN)^2$, and by subtracting this from the total number of elements in the matrix γ for two sequences of length N we obtain:

$$N^2 - (N - rN)^2 = (2r - r^2) \quad (2.3)$$

Using Equation 2.3, the ratio of the number of computations using the S-C band versus Classic DTW is:



(a) Approximate theoretical speed-up factor for the S-C band as r is increased.

(b) Approximate theoretical speed-up factor for the Itakura band as m_1 is increased.

Figure 2.3: Approximate theoretical speed-up factors for both constraint region optimisations as their respective hyperparameters are increased.

$$\frac{N^2(2r - r^2)}{N^2} \quad (2.4)$$

The speedup factor is then proportional to the inverse of this ratio:

$$(2r - r^2)^{-1} \quad (2.5)$$

From this relation it is clear that the speedup factor decays exponentially with respect to r , as illustrated in Figure 2.3a. Where incremental changes in r for lower values have a much larger impact on runtime.

2.2.2. Itakura parallelogram

Unlike the S-C band, the Itakura parallelogram (or Itakura band) was proposed solely as a method to speed up the DTW computation [12]. This is accomplished by introducing a global constraint region R that substantially reduces the total number of elements computed in γ . Illustrated in Figure 2.2b, this is accomplished by constructing four lines, and setting R to the parallelogram shaped region that is enclosed by these lines. In general, only the gradient of l_1 (m_1) is set, after which the gradients of l_2 , l_3 , and l_4 are derived as follows (for to sequences of length N): $m_2 = m_3 = \frac{1}{m_1}$ and $m_4 = m_1$. The intercepts of each line are set such that the corners of the parallelogram overlap $\gamma(1, 1)$ and $\gamma(N, M)$,

as depicted. These line equations are given below:

$$l_1(x) = m_1x + 1 \quad (2.6a)$$

$$l_2(x) = m_2x - m_2 \quad (2.6b)$$

$$l_3(x) = m_3x + N(1 - m_3) + m_3 \quad (2.6c)$$

$$l_4(x) = m_4x + N(1 - m_4) - 1 \quad (2.6d)$$

To determine the number of elements enclosed in the parallelogram, we start by first determining the intercept points between l_1 and l_3 , and l_2 and l_4 . This is accomplished by first equating Equations 2.6a and 2.6c and solving for x :

$$x_1 = \frac{N(1 - m_3) + m_3 - 1}{m_1 - m_3} \quad (2.7)$$

By noting that $m_3 = \frac{1}{m_1}$, this equation can be further simplified to:

$$x_1 = \frac{N - 1}{m_1 + 1} \quad (2.8)$$

Substituting Equation 2.8 into Equation 2.6a, the y-coordinate is determined as:

$$y_1 = \frac{N \cdot m_1 + 1}{m_1 + 1} \quad (2.9)$$

Due to the symmetrical nature of the parallelogram, the intercept point of l_2 and l_4 (x_2, y_2) can be determined by simply swapping x_1 and y_1 , the intercept of l_1 and l_3 .

Now the area inside the parallelogram can be calculated. The Euclidean distance between the two intercept, a , is:

$$\begin{aligned} a &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ &= \sqrt{\frac{2N^2m_1^2 + 8Nm_1 + -4N^2m_1 + 2N^2 + 8 - 8N}{m_1 + 1}} \end{aligned} \quad (2.10)$$

The length of the diagonal (b) between the other two intercepts is simply $\sqrt{2}N$. The total area enclosed by the parallelogram can be calculated as:

$$\begin{aligned}
A &= \frac{a \cdot b}{2} \\
A &= \frac{1}{2} \cdot \sqrt{\frac{2N^2m_1^2 + 8Nm_1 - 4N^2m_1 + 2N^2 + 8 - 8N}{m_1 + 1}} \cdot \sqrt{2}N
\end{aligned} \tag{2.11}$$

Assuming $N^2 \gg N$ this equation can be further simplified:

$$A \approx \frac{N^2}{2} \sqrt{\frac{4m_1^2 - 8m_1 + 4}{m_1 + 1}} \tag{2.12}$$

This relation approximately describes the number of elements computed for varying values of m_1 , and the inherent reduction computation time. Following the same approach in Equation 3.1, the speed up factor for the Itakura band is:

$$\left(\frac{1}{2} \sqrt{\frac{4m_1^2 - 8m_1 + 4}{m_1 + 1}} \right)^{-1} \tag{2.13}$$

This is further illustrated in Figure 2.3b, where it is clear that the approximate speed-up factor decays exponentially as m_1 is increased.

2.2.3. FastDTW

FastDTW is a multilevel approach to find constraint regions that will still allow for the optimal warping path to be found. Unlike the S-C and Itakura bands, these constraint regions, in general, differ for each temporal sequence pair. This is accomplished by downsampling each temporal sequence to a very low resolution, computing the DTW, and projecting the resulting warping path to incrementally higher resolutions, each time using the warping path computed at the lower resolution as the constraint region when computing the higher resolution DTW [1].

Figure 2.4 shows the four resolutions that are considered for two temporal sequences of length 16. The optimal warp path P^* (denoted by the dark line) determined at the lowest resolution is projected into the next higher resolution, P'^* (dark grey cells). This process is repeated until the native resolution is evaluated. However, it is still possible that the optimal warping path may not be contained in P'^* . To ensure the optimal warp path is found at each iteration, an additional *radius* parameter is introduced, which controls the additional number of cells outside P'^* to be considered (light grey cells).

Since only the cells in the *neighbourhood* of P'^* are computed, the length of which grows linearly with respect to the length of the temporal sequences, the algorithm has an approximate $O(N)$ time complexity. However, this only holds for sufficiently large sequences, such that the extra computation required to create the multiple resolutions

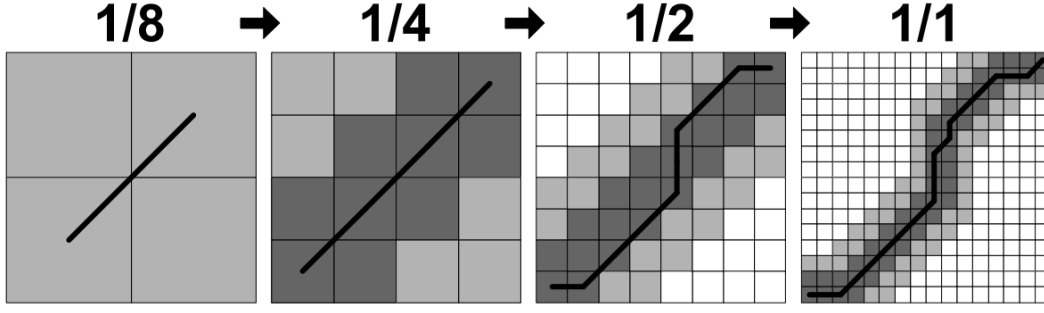


Figure 2.4: The four different resolutions evaluated during a complete run of the FastDTW algorithm on two temporal sequences of length 16. Dark line indicates the current warping path, dark grey cells show the expanded P'^* and the light grey cells show the additional cells considered as per the *radius* parameters. [1]

is outweighed by the total number of distance computations performed to compute the matrices D and γ .

2.2.4. Parallelisation

CPU multi-threading

The most common DTW parallelisation technique is multi-threading. Multiple central processing unit (CPU) cores are utilised to compute the DTW between different temporal sequences in parallel [13] [14]. Whilst this is a viable technique, the factor by which these computations can be sped up is directly proportional to the number of CPU cores available. Thus this approach is attractive when one has access to large computing clusters, however, these are generally expensive and inaccessible.

GPU parallelisation

Another more sophisticated approach is to diverge from the classical method of populating γ row by row, and take advantage of the diagonal data dependency of γ . Illustrated in Figure 2.5, the computation of each element in a diagonal is independent of all other elements in that same diagonal, and only requires elements from the last two diagonals to determine its value. This means that if the matrix γ is populated in this rolling diagonal fashion, one can take advantage of the aforementioned Independence and compute each element of each sequential diagonal in parallel [15]. However, for realistically large temporal sequences, this would require too many CPU cores to be viable. GPUs, on the other hand, which can contain thousands of processing cores, are a viable option for this parallelisation approach [16]. Even so, to utilise the GPU it is required to constantly load data from the CPU on to the GPU and back. This process is a known bottleneck, and as such, the speed increase from computing DTW in this way will only be realised for sufficiently large sequences, for which the time lost in data transfer is outweighed by the benefits of parallelisation.

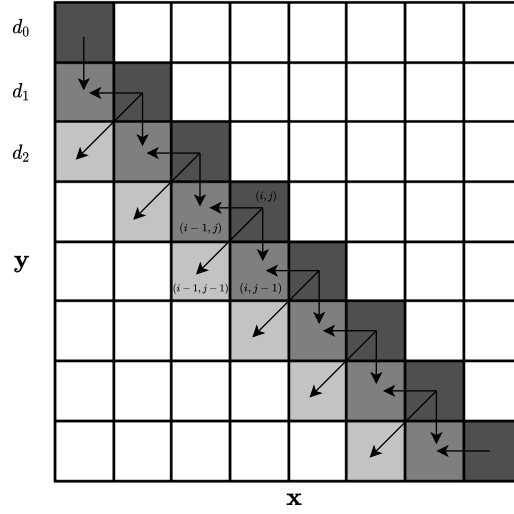


Figure 2.5: An illustration of how to populate γ along the diagonals. Arrows show dependencies. The diagonals shaded from dark to light are denoted as d_0 , d_1 , and d_2 respectively. Elements along d_0 can be computed in parallel.

2.3. DTW Keyword Spotter

In low resource settings, traditional keyword spotting (KWS) based on ASR techniques are not feasible. In these situations, DTW proves to be an appropriate detection technique. The process by which this is accomplished is detailed below.

As previously mentioned, DTW can be used to align two temporal sequences $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_M]$ where \mathbf{x}_i and $\mathbf{y}_j \in \mathbb{R}^d$. In KWS where the objective is to determine whether the j^{th} repetition of the i^{th} keyword template $\mathbf{X}_{i,j}$ is present in an utterance \mathbf{Y} , the template can be slid over \mathbf{Y} , each time computing the DTW cost $c_{i,j,q}$ for each window of overlap $k = 1, 2, \dots, K$, where q is the starting index of the window of overlap in \mathbf{Y} . This concept is shown in Equation 2.14:

$$c_{i,j,q} = \text{DTW}(\mathbf{X}_{i,j}, \mathbf{Y}(q, q + N - 1)) \quad (2.14)$$

the current KWS implementation uses cosine distance as the distance function d , where $d(\mathbf{x}_i, \mathbf{y}_j) \in [0, 2]$. The resulting DTW cost can then be used as a measure of similarity (J) between $\mathbf{X}_{i,j}$ and $\mathbf{Y}(q, q + N - 1)$. To achieve this, c is first normalised by the size of the matrix such that $c \in [0, 2]$. Then, this converted to a similarity score $J \in [0, 1]$. Where a value of 1 indicates an identical pair, and values approaching 0 indicate a dissimilar pair. This process is detailed below:

$$J_{i,j,q} = -\frac{1}{2} \cdot \frac{c_{i,j,q}}{N^2} + 1 \quad (2.15)$$

Note that the window of overlap corresponds to the length of the keyword, N , and a frame skip, ζ , of 3 is used. This process is repeated for every keyword. Then, to decide if a

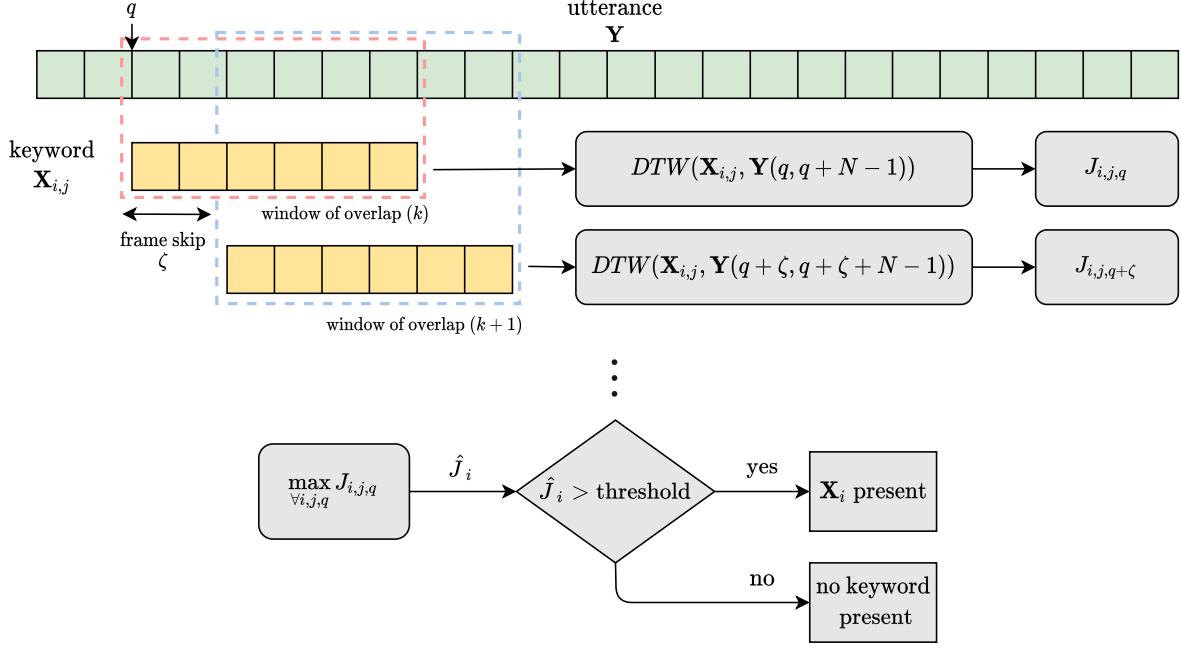


Figure 2.6: An illustration of the DTW keyword spotting architecture. For each window of overlap the DTW is computed and a similarity score J is produced, this process is then repeated for each repetition for each keyword. Once this is complete the largest similarity score is determined and if it is larger than a predetermined threshold, it is decided that the utterance contains the corresponding keyword.

keyword is contained in the utterance, the largest value for all the DTW similarity scores, $\hat{J}_i = \max_{\forall i,j,q} J_{i,j,q}$, is compared to an appropriate threshold and a decision is made. As this concept is important to grasp, and is used extensively in later chapters, it is illustrated in Figure 2.6 for more clarity.

2.4. Acoustic Feature Vectors

In most audio recordings, unwanted attributes are present. These include speaker stressing, background noise, amplitude variations, speaker emotion, pronunciation and more [17]. These variations in recordings contribute to a lack of speech intelligibility and degrade the performance of speech recognition systems [18]. To separate the desired information from some of these unwanted attributes and to uncover information not initially evident in the audio (such as frequency content), acoustic feature vectors are used to represent segments of audio.

Whilst a plethora of acoustic feature vectors exist, in this application only mel frequency cepstral coefficients (MFCCs) are considered. This is to maintain consistency with previous work on this topic [4] [11].

2.4.1. Mel frequency cepstral coefficients

Mel frequency cepstral coefficients (MFCCs) are used as acoustic feature vectors. MFCCs capture spectral information, whilst maintaining a low dimensionality by only capturing the perceptually important characteristics of speech [19].

This is accomplished by first computing the Short Time Fourier Transform (STFT) of the audio signal, with a frame length of 10ms – 50ms. A mel filter bank is then applied to each frame. The filter bank consists of 26 triangular band-pass filters for which the bandwidth and spacing is depended on the mel scale, illustrated in Figure 2.7. Each filter outputs the weighted sum of its associated spectral energy components, S_k . The logarithm of these values are then transformed back to the time domain using the Discrete Cosine Transform (DCT) which outputs 13 cepstral coefficients, shown in Equation 2.16, where K is the number of filter banks [20]:

$$C[m] = \sum_{k=0}^{K-1} \log_{10}(S_k) \cos\left(\frac{\pi m(k - 0.5)}{K}\right), \quad m = 0, \dots, 12 \quad (2.16)$$

The resulting 13-dimensional acoustic feature vector is named the Mel Frequency Cepstral Coefficients. To capture the change of these features over time, the first and second derivatives of these cepstrums (velocity and acceleration) are also appended which are approximated using a predetermined number preceding frames as well as following frames, resulting in a 39-dimensional acoustic feature vector for each frame [21]. To calculate the derivatives of the cepstrums, the following formula is used:

$$d_t = \frac{\sum_{n=1}^N n(c_{t+n} - c_{t-n})}{2 \sum_{n=1}^N n^2} \quad (2.17)$$

Where d_t is the velocity coefficient for the cepstral coefficient c_t of frame t computed in terms of the neighbouring coefficients from frame $t - N$ to $t + N$. Typically, the value for N is 2. The same process is followed when calculating acceleration, replacing the cepstral with velocity coefficients.

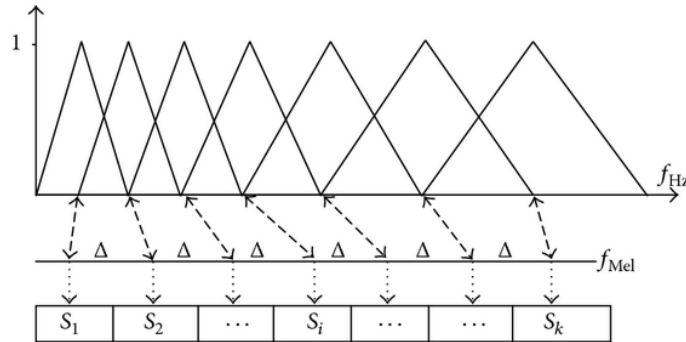


Figure 2.7: Spectrum of Mel filter banks and the corresponding Mel frequency mapping [2].

2.5. Chapter summary

In this chapter we detailed the fundamental concepts of DTW and provided a brief overview of some recent applications of DTW to solve real-world problems. We then presented and explained some popular existing DTW optimisations, those being: the S-C band, the Itakura band, FastDTW, multi-threading, and GPU parallelisation. Since the concept of DTW KWS is used extensively in later chapters, we provided an in-depth explanation. Lastly, the importance of acoustic feature vectors was highlighted, and a description of the ones chosen for this application, MFCCs, was given. In the next chapter will present and describe new DTW optimisations formulated by us specifically for KWS.

Chapter 3

Proposed optimisations for keyword spotting

This chapter describes new DTW optimisations formulated specifically for the KWS application shown in Figure 2.6. While the optimisations described in Section 2.2 focused on general computation reduction, the focus is now shifted to take advantage of the specific keyword spotter architecture. The proposed optimisations include cost matrix caching, dynamic global constraint regions, and adjustable frame skip. The proposed methods are evaluated in Chapter 5.

3.1. Cached D DTW

In the baseline DTW keyword spotting architecture as shown in Figure 2.6, as each instance the keyword is shifted over the utterance, the entire distance matrix D_k is computed for each window of overlap k . However, since the keyword is only shifted by a known frame skip ζ which is always smaller than the number of frames in the keyword, a large portion of D_k can be cached and reused when calculating D_{k+1} since most of the utterance frames contained in the next window of overlap are the same. This concept is illustrated in Figure 3.1.

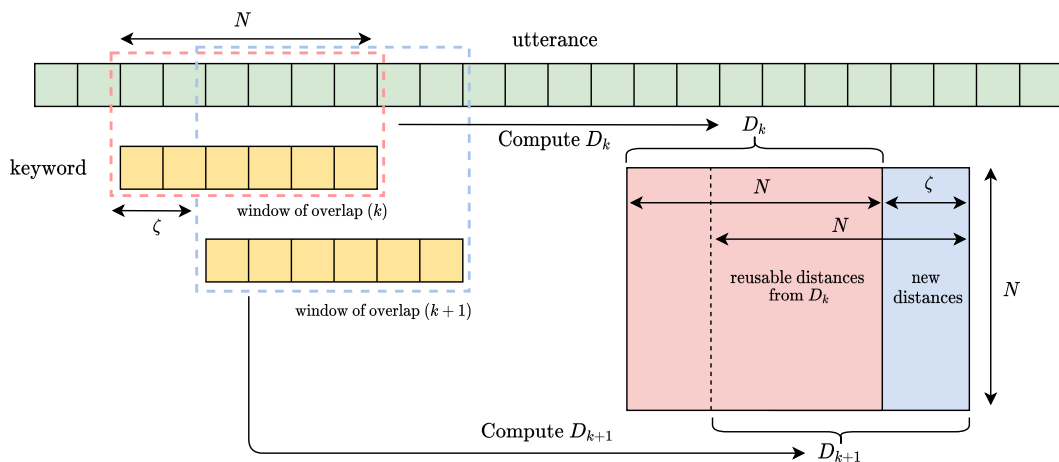


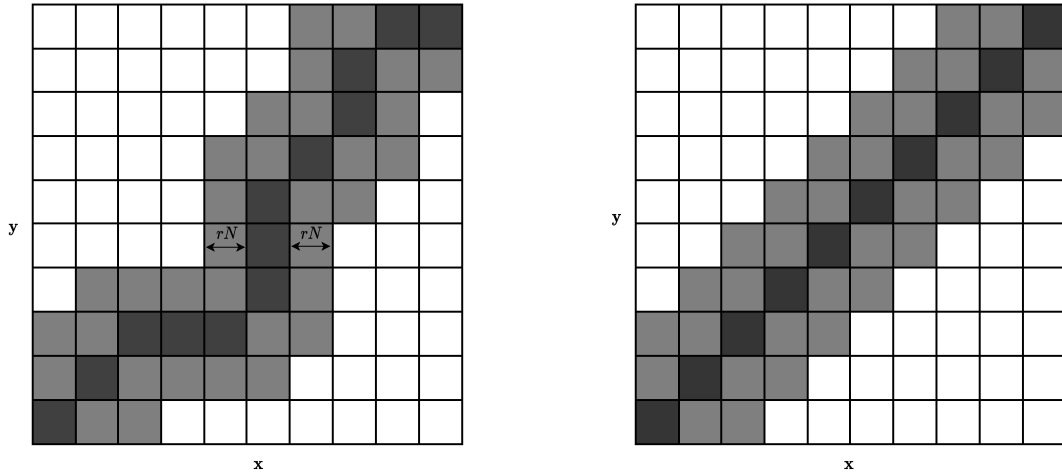
Figure 3.1: A high level diagram illustrating the key concepts of Cached D DTW.

Whilst computing D is not the only source of DTW's computational complexity, it is substantial and a reduction will decrease runtime. Assuming an average window of overlap of 90 frames, and a frame skip of 3, the optimisation results in 96.7% fewer distances calculated each time the keyword is shifted over the utterance. Whilst the time complexity remains $O(N^2)$, the absolute number of computations is strongly reduced.

3.2. P^* DTW

When the frame skip is small, one might assume that the optimal warping path, P^* , would not change substantially between successive DTW alignments, since the majority of the information contained in the window of overlap remains the same. Thus, borrowing from the fundamental concept of FastDTW, an expanded P_{old}^* can be used as a global constraint region, R , for the next DTW calculation. This is illustrated in Figure 3.2a, where r is the expansion radius. In this application, r is expressed as a fraction of the length of the temporal sequences such that $r \in [0, 1]$. Thus the width of the band illustrated in Figure 3.2a becomes rN . This optimisation has the potential to substantially reduce the time complexity of DTW, should an appropriate value for r be chosen.

This reduction is further demonstrated by an approximate theoretical analysis. Given a keyword length of N , and the warping path illustrated in Figure 3.2b, the total number of additional computations on either side of P^* are $2N^2r$. The global constraint region thus contains a total of $4N^2r + N$ cells. If r is chosen to be sufficiently small, such that $4N^2r \ll N$, the time complexity approximately becomes linear i.e. $O(N)$.



(a) Illustration of the constraint region R (shaded in grey) that will be imposed by the P^* optimisation on the next alignment. Dark grey indicates P^* .

(b) Warping path used to derive the approximate runtime benefit of P^* DTW.

Figure 3.2: Illustrations of the global constraint regions imposed by P^* DTW for temporal sequences of length N .

3.3. Grad DTW

In the current implementation of the DTW keyword spotter, the frame skip is kept constant. We now consider whether it can be advantageous to dynamically change the frame skip throughout the computation of the DTW costs between a keyword-utterance pair. This could for example be beneficial if the frame skip can be adjusted according to on a given region of the utterance’s likelihood of giving good DTW scores. In this optimisation the gradient estimated from the last two DTW costs, Δ_c , is used to determine the frame skip for the next DTW computation. We make an informed assumption that if the DTW costs are decreasing, we are nearing a region in the utterance that is similar to the keyword, and if the DTW costs are increasing, we are moving towards a dissimilar region. This simple intervention allows the frame skip to reduce when Δ_c is negative (DTW costs are decreasing) and to increase when Δ_c is positive (DTW costs are increasing). This allows the keyword spotter to *move faster* through regions where the utterance and keyword are dissimilar, and *slow down* when a possible match seems to be approaching. By setting appropriate lower and upper thresholds of this adaptive frame skip, a substantial number of DTW computations can be skipped, reducing overall runtime. This optimisation is named gradient (Grad) DTW.

In a naive attempt to quantify the speed increase that Grad DTW provides, we assume that the costs increase and decrease with a uniform distribution and thus the frame skip has an equal probability of being ζ_{\min} or ζ_{\max} . By stating that runtime is inversely proportional to the mean frame skip, we can begin to derive the approximate runtime relation provided by Grad DTW.

For classical DTW, the frame skip is fixed to ζ_{\min} and therefore:

$$\text{runtime} \propto \frac{1}{\zeta_{\min}} \quad (3.1)$$

However, for Grad DTW, considering the aforementioned assumptions the runtime frame skip relation can be determined as shown below, where k is the factor by which ζ_{\max} is greater than ζ_{\min} :

$$\begin{aligned} \text{runtime} &\propto \frac{1}{0.5 \cdot \zeta_{\min} + 0.5 \cdot k \cdot \zeta_{\min}} \\ \text{runtime} &\propto \frac{1}{0.5(1+k)\zeta_{\min}} \end{aligned} \quad (3.2)$$

Thus, the approximate theoretical speed increase factor provided by Grad DTW is $0.5(1+k)$. If an appropriate value for k is chosen, and Grad DTW is coupled with other optimisations, this factor can be further increased which will be demonstrated in later chapters. Since in practice the DTW costs may not be uniformly distributed, Equation 3.2 will have to be tested by empirical evaluation.

3.4. GRAM DTW

One limitation of Grad DTW is that the frame skip will be reduced towards the lower threshold immediately when the DTW cost begins to decrease. However, the DTW cost was observed to fluctuate substantially when no keyword is present in the window of overlap. On the other hand, when there is a keyword present, the DTW cost was observed to consistently decrease before it reaches the minimum. These two behaviours are illustrated in Figure 3.3. Ideally, the keyword spotter should move quickly through the regions where the score is fluctuating, and then gradually slow down as it moves closer to the region where a keyword is present.

To address this issue, a common machine learning tool is appropriated: momentum [22]. In this implementation, momentum is used to compute an exponentially weighted average of DTW cost gradients, which is then used to determine an appropriate frame skip. This method is named Gradient and Momentum (GRAM) DTW. The classical momentum algorithm is modified to suit this use case and is shown in Equation 3.3, where β is the momentum hyperparameter, ζ is the frame skip, and k is used to scale Δ_c to appropriate values to adjust ζ . Naturally, maximum and minimum thresholds are still imposed on ζ .

$$\zeta_{\text{new}} = \text{round}(\beta \cdot \zeta_{\text{old}} + \Delta_c(1 - \beta)k) \quad (3.3)$$

Since each keyword-utterance pair and the associated DTW costs are unique, it is hard to predict the speed increase this method may yield. However, as a rough estimation, (considering the data in Figure 3.3) one can assume that about 90% of the costs are associated with regions that do not contain a keyword. Due to the fluctuating nature of these regions where no keyword is present, if the optimisation performs as expected it should move through these regions at considerable speed (towards ζ_{max}). If we assume the average frame skip through these regions is k times greater than the lower threshold,

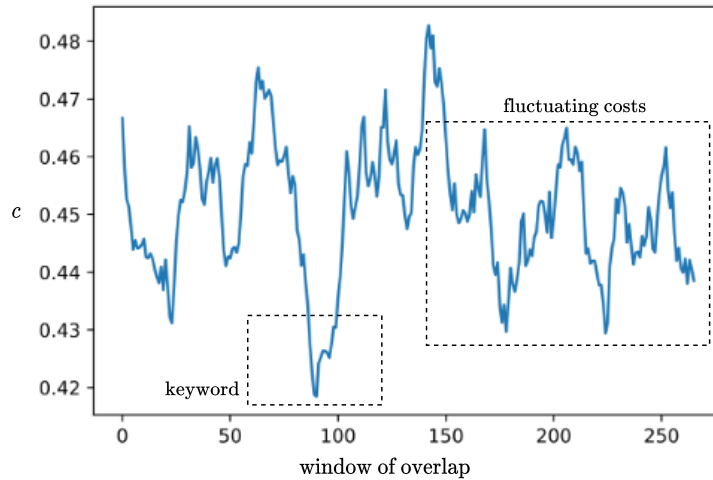


Figure 3.3: An example illustrating the DTW costs (c) as a keyword is approached, and how the DTW cost fluctuates when there is no keyword present in the window of overlap.

ζ_{\min} , (the value the frame skip should be near a keyword) and by stating that runtime is inversely proportional to the mean of ζ , the potential speed increase can be quantified. This is demonstrated below:

$$\begin{aligned} \text{runtime} &\propto \frac{1}{0.1 \cdot \zeta_{\min} + 0.9 \cdot k \cdot \zeta_{\min}} \\ \text{runtime} &\propto \frac{1}{(0.1 + 0.9 \cdot k)\zeta_{\min}} \end{aligned} \tag{3.4}$$

It has thus been shown that if the aforementioned assumptions hold, one can expect a speed increase factor of approximately $0.1 + 0.9 \cdot k$ when using GRAM DTW. Since k is partially dependant on ζ_{\max} , if an appropriate ζ_{\max} is chosen and this optimisation is coupled with others, the speed-up factor can be further increased. This will be tested in later chapters.

3.5. Chapter summary

In this chapter we proposed four new optimisations, namely: Cached D DTW, P^* DTW, Grad DTW, and GRAM DTW. Each optimisation utilises the architecture of the keyword spotter to achieve a runtime speed-up. We presented the derivations of the approximate theoretical runtime benefits for each, which will be further tested by empirical evaluation in later chapters. The next chapter presents our experimental setup, and will also detail our evaluation metrics, data set, and hyperparameter selection.

Chapter 4

Experimental setup

This chapter discusses the implementation of the DTW optimisations described in Chapters 2 and 3. The selection of hyperparameters for each optimisation is also described. These are tuned using a development data set. To ensure clarity, we start by explaining our evaluation metrics, the data set used and our architecture.

4.1. Evaluation metrics

Whilst the goal remains to decrease the inference speed of DTW for keyword spotting, it is important to evaluate each optimisation with standard classifier metrics that will highlight its performance. All evaluation metrics were implemented using the `scikit-learn` python package [23].

To better understand these metrics, we start by briefly outlining the fundamental operation of our classifier. The keyword spotter computes DTW costs for each window of overlap for each keyword utterance pair. These costs are then converted to a similarity score $J \in [0, 1]$, where 0 indicates no keyword present in the window of overlap and values approaching 1 indicate a greater likely-hood of a keyword being present. Next, for each keyword we take the highest similarity score (best) of all those computed over the whole utterance, and use that score as an indicator that the utterance containing the keyword. After obtaining the best of these utterance-level scores for each keyword, we choose an appropriate decision threshold – which will dictate the label we give each utterance. Since We are only searching for one keyword in this application, the response of the classifier is binary (either the keyword is present or not). Since each utterance is transcribed, we can obtain the ground truth labels for each. In this application, we label utterances containing the keyword with a 1 and those that do not with a 0.

By comparing the outputs of the classifier with the ground truth labels of the data we can infer how well the classifier is performing, given the chosen decision threshold. This is often evaluated with four basic metrics: the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) decisions. These are commonly illustrated by confusion matrices, as shown in Figure 4.1.

		Predicted label	
		1	0
Ground truth label	1	TP	FN
	0	FP	TN

Figure 4.1: A confusion matrix depicting the ground truth and predicted labels for TP, TN, FP, and FN decisions. Cells highlighted in green indicate metrics where higher values indicate a better classifier, whereas red indicates metrics where lower values infer better performance.

4.1.1. Receiver operating characteristic

The receiver operating characteristic (ROC) curve is achieved by plotting the true positive rate (TPR), where $TPR = \frac{TP}{TP+FN}$, on the y-axis against the false positive rate (FPR), where $FPR = \frac{FP}{FP+TN}$, on the x-axis as the decision threshold is varied. This is a useful visualisation tool since the trade-off between TPR and FPR can be easily identified. A curve that passes through the upper left corner of the plot indicates a perfect classifier, whereas a curve that lies along the 45° diagonal represents a classifier making random decisions. This metric is mainly used to visualise system performance and was not used when selecting hyperparameters.

4.1.2. Area under the curve

While ROC is a useful visualisation tool, it can become difficult to use effectively when comparing classifiers of similar performance. For this scenario, it is useful to compare classifiers is the area under the ROC curve (AUC). AUC is computed by determining the total area under the ROC curve and is used as a single metric that characterises the classifier across all operating points. Note, $AUC \in [0, 1]$ where higher values indicate a better classifier and values approaching 0.5 indicate performance similar to a random classifier.

4.1.3. Equal error rate

The equal error rate (EER) is the error rate at the decision threshold where the false-negative rate (FNR) is equal to the FPR. As the decision threshold of a system is increased,

the FPR will naturally decrease while the FNR will increase. For a good classifier, the FPR should decrease rapidly, and thus FNR and FPR should cross over at a low error rate, depicted in Figure 4.2. Thus, a lower EER indicates a better classifier.

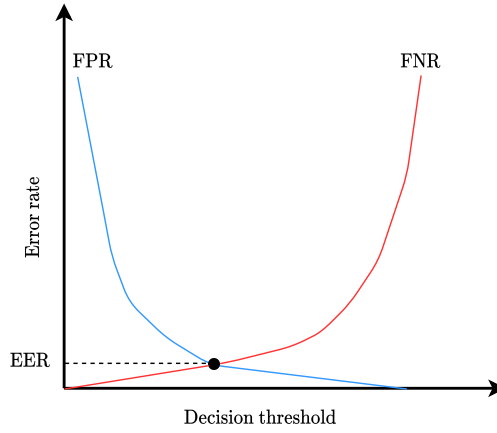


Figure 4.2: Representation of FPR and FNR as the decision threshold is increased. The error rate where FPR and FNR cross is labelled the EER.

4.1.4. Runtime and speed-up factor

With the goal of producing optimisations that reduce the inference speed of classical DTW for keyword spotting, the total time it takes to compute all the DTW costs are reported. This is also expressed by a speed-up factor, which is the relative speed increase the optimisation provides compared to classical DTW.

4.2. Data

As highlighted in Section 2.3, DTW is used to determine whether keywords are present in a set of utterances. While in practice the contents of the utterances are unknown, for development purposes, a transcribed English corpus of South African Broadcast News (SABN) is used. Whilst it is common in machine learning to have a training set, this application does not require one. Instead, the data is split into a development set which is used to tune hyperparameters and a test set which is used to evaluate the performance of each optimisation on a large amount of unseen data.

4.2.1. Keywords templates

The current implementation of the keyword spotter makes use of 40 different keywords, each spoken as isolated words twice by 24 individuals (12 male and 12 female) [11]. This results in a total of 1920 keywords. Due to computational constraints, and to simplify the analysis, a subset of only one of these keywords is used to evaluate the DTW optimisations.

Due to a large number of occurrences in the SABN corpus, the keyword *Government* is chosen. From the 24 recordings, 12 were selected at random to make up the keyword template subset for experimentation.

4.2.2. SABN corpus

The SABN corpus is comprised of 23 hours of transcribed speech from news bulletin broadcasts between 1996 and 2006 by a South African radio station, SAFM. the corpus contains a mix of newsreader speech, interviews, and crossings to reporters from both male and female speakers [24].

As previously outlined, a subset of the corpus is used for development whilst another larger and completely different subset is reserved as a test set. Both subset's have around a 1 : 6 split between utterances containing a keyword and utterances that do not. This is detailed in Table 4.1:

Table 4.1: The SABN subsets generated for development and testing, indicating the the number of utterances that contain a keyword (kw) and those that do not. The total duration is also given in hours (h).

Set	kw ✓	kw ✗	Total	Duration
Dev	40	200	240	0.5h
Test	80	400	480	1h

4.2.3. MFCCs

Both the keywords and utterances exist as `.wav` files, a standard file format for audio. These raw audio files are converted to the 39-dimensional MFCCs as described in Section 2.4.1. To maintain consistency with the current work on the keyword spotter being conducted by the DSP lab, these MFCCs were supplied and not recomputed.

4.3. Architecture

All work was implemented in the Python 3.7 environment. This was chosen for two reasons: (a) Python allows for quick prototyping due to its simplistic and intuitive nature, and (b) past work on this subject was implemented in this environment, and as such it was chosen for seamless integration.

All optimisations were written in a single python `.py` file, which was then imported into a `jupyter notebook` where the experiments were conducted. Experiments were executed on a PC with an Intel i5 6-core processor and 16GB of RAM.

4.3.1. Just-In-Time compiler

Unlike traditional programming languages like C or C++, python uses an interpreter and is not compiled. This allows the language to be dynamically typed. However, it also means that code is executed slowly since each time a piece of code is run, it must be reinterpreted. This is not ideal for an application such as this one, of which the main goal is to reduce runtime. To circumvent this issue a Just-in-Time (JIT) compiler is used. A JIT compiler compiles the source code into machine code, which is optimised for the CPU architecture making the execution time much faster. Selected functions are given the `@jit` decorator so that when they are called for the first time they are compiled into machine code. This is accomplished with the Numba python package [25].

4.4. Keyword spotter

A modular keyword spotter was built, such that each optimisation could be tested independently. First, an empty dictionary data structure is created. This will serve essentially as a simple *database* in which to store the associated DTW costs for each keyword utterance pair. For each utterance, a new key is added to the dictionary (the file name of the utterance), which then points to a new nested dictionary which is populated with keys for each keyword. Each one of these keys points to a list, which is used to store the DTW costs for every window of overlap. The pseudo code is given below:

Algorithm 4.3: Keyword spotter algorithm

```

costs  $\leftarrow$  {}
for each Y in utterances do
    costs["utt_fname"] = {}
     $M \leftarrow \text{len}(\mathbf{Y})$ 
    for each X in keywords do
        costs["utt_fname"]["kw_fname"] = [ ]
         $q \leftarrow 0$ 
         $N \leftarrow \text{len}(\mathbf{X})$ 
        while  $q \leq M - N$  do
             $c = \text{DTW}(\mathbf{X}, \mathbf{Y}(q, N - 1))$ 
            costs["utt_fname"]["kw_fname"].append( $c$ )
             $q \leftarrow q + \zeta$ 
        end while
    end for
end for

```

Upon completion, this dictionary is further processed, whereby each cost is converted to a similarity score. Then, the maximum similarity score for each utterance is determined, which is stored in a similar nested fashion. To quantify the performance of the specific optimisation, these values can be compared with the ground truth labels for each utterance (1 if it contains the keyword and 0 if it does not). This comparison is performed using the previously described metrics.

4.5. Baseline

To make an informed decision when selecting hyperparameters for the optimisations, a frame of reference must be established. Classic DTW was implemented and run on the development set, and the results are presented in Table 4.2.

Table 4.2: Baseline results for classic DTW on the development set. Values are expressed as percentages (%), i.e. an AUC of 100% represents an area under the curve of 1.

AUC	EER	Time (min)
68.13	34.67	6.0

As part of the previous work done in [11], a public repository was created which includes a python implementation of the DTW keyword spotter. To ensure we had correctly implemented the system, we compared the results generated by this existing system to ours on the development set. Both systems achieved identical EER and AUC scores, and likewise, the ROC plots were the same as illustrated in Figure 4.3.

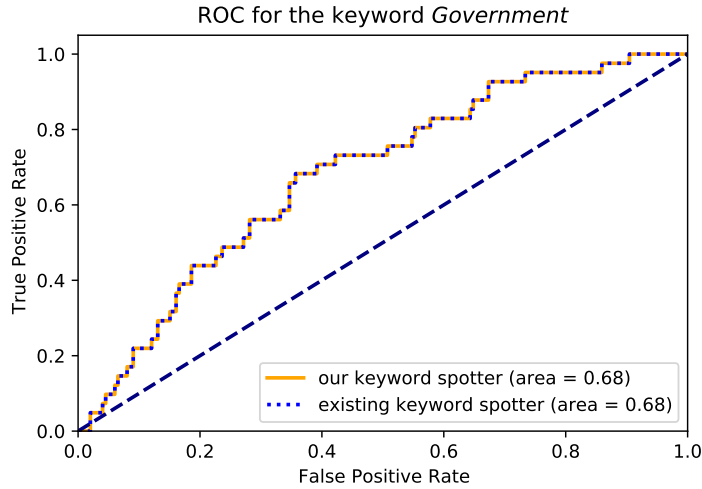


Figure 4.3: ROC plots for both the existing DTW keyword spotter and our implementation.

4.6. Implemented optimisations

A high-level description with regards to the implementation of each optimisation is given below. For optimisations that require hyperparameter selection, an in-depth explanation on the methods used to find the optimal hyperparameter values is also given. Observations during hyperparameter selection are also stated. It is important to note that all hyperparameter selection was performed using only the development set.

Not all the optimisations detailed in Chapter 2 were implemented for various reasons. FastDTW was not considered as it was observed to be slower than classic DTW in initial

testing. It is suspected that creating the additional resolutions was too computationally expensive compared the reduction in computations the constraint region provided. CPU multi-threading is already used in the existing DTW implementation and further investigation was therefore deemed unnecessary. Some investigation was done with regards to GPU parallelisation. However, the cost of loading such (relatively) small matrices onto the GPU and back to the CPU proved to outweigh any parallelisation benefits. This initial testing of GPU parallelisation was accomplished using a publicly available python implementation provided by the authors of [16].

4.6.1. S-C band

In this implementation, we construct two lines which define the boundary of the S-C band. Then, we use a conditional statement inside the inner loop of Algorithm 2.1 which only updates γ if (i, j) falls within these two lines, thus reducing the number of unnecessary computations.

Hyperparameter selection

As described by Equation 2.3, the number of computed cells grows exponentially with respect to r . However, making r too small might result in the optimal warping path being missed, and the corresponding DTW cost being inaccurate. This will subsequently degrade the performance of the keyword spotter.

AUC and EER is plotted against r and are shown in Figure 4.4. After analysing the results, it was clear r could be decreased significantly before any performance impact was observed. For this application, this makes sense because the warping path of two similar temporal sequences will always be close to the diagonal of the matrix γ . Even if the width of the band is small, these warping paths are still usually captured, and the subsequent true DTW distance is still computed and similarity scores close to those produced with

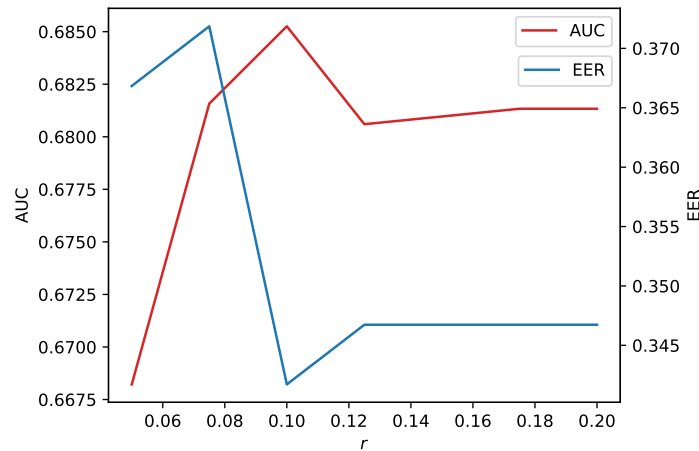


Figure 4.4: AUC and EER on the development set for increasing values of r for the S-C band DTW optimisation.

classical DTW will still be observed. Likewise, dissimilar sequences will result in large DTW distances, as the small bandwidth would not contain the ideal warping path. Thus the inherent quality of DTW that is fundamental to the keyword spotter is still intact.

A value of $r = 0.1$ was chosen as it was observed to be the value where AUC and EER were the best. Interestingly, the values for AUC and EER were slightly better than those produced by the baseline.

4.6.2. Itakura band

We follow a similar method as described previously for the S-C band, and construct the four lines that define the Itakura band. Then, we place a conditional in the inner loop of Algorithm 2.1 which will only update γ if (i, j) lies within the region these lines construct.

Hyperparameter selection

Whilst Equation 2.7 is a little more difficult to interpret, it is clear that there is a non-linear relation between m_1 and the number cells that are computed and by extension the runtime. The method followed to find the optimal value to m_1 is the same employed in Section 4.6.1. Development set AUC and EER scores are plotted for increasing values of m_1 , depicted in Figure 4.5.

For reasons similar to those described in Section 4.6.1, even small constraint regions ($m_1 = 1.1$) had decent metric scores. Once again, there was a region in the plot where both AUC and EER were better than the baseline. $m_1 = 1.35$ corresponded to the region where the highest value for AUC was obtained and the lowest value for EER was observed and was therefore chosen.

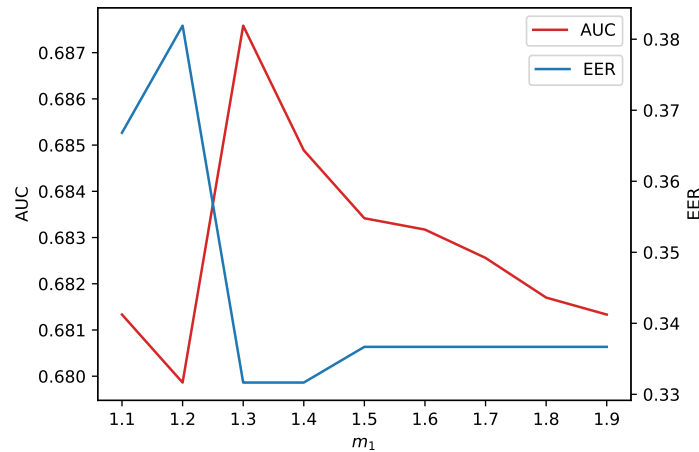


Figure 4.5: AUC and EER on the development set for increasing values of m_1 for the Itakura band DTW optimisation.

4.6.3. P^* DTW

To create the constraint region required for P^* DTW, the warping path of the previous alignment was iterated through, generating a list of indices for the expanded region of γ . Subsequently all elements of the matrix γ not within this list were flagged by assigning them a value of ∞ . Finally, a conditional was placed inside the inner loop of Algorithm 2.1, which only updated γ if $\gamma(i, j)$ did not equal ∞ .

Hyperparameter selection

To ensure the approximate linear time complexity relation is maintained, which was highlighted in previous sections on P^* DTW, the value for r needs to be chosen to be sufficiently small. A similar approach to previous hyperparameter tuning is followed, and the plot for AUC and EER scores on the development set as r is increased is shown in Figure 4.6.

The plot highlights that to achieve performance similar to classic DTW, a trade-off is made between performance and speed. The larger r , the larger the constraint region, which subsequently increases runtime. Whilst $r = 0.125$ does not result in metrics equal to the baseline, it is the largest value for r before P^* DTW's runtime was observed to surpass that of Classic DTW. This is due to the additional computations required to set up the constraint region, which begin to overshadow the speed benefits the constraint region provides.

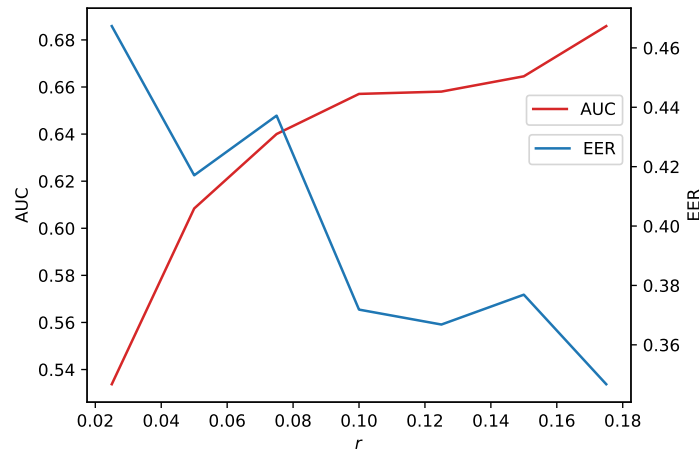


Figure 4.6: AUC and EER on the development set for increasing values of r for the P^* DTW optimisation.

4.6.4. Cached D DTW

To implement cached D DTW, the entire cost matrix is computed only in each first iteration of the inner loop in Algorithm 4.3, thereafter which it is cached for future iterations. In each subsequent iteration, ζ is used to determine the number of new distances that must

be computed. These new distances are then appended to the reusable portion of the cached D matrix ($N \times N - \zeta$), resulting in the new D matrix ($N \times N$).

4.6.5. Grad DTW

To implement Grad DTW, the first iteration of the `while` loop in Algorithm 4.3, ζ was initialised to ζ_{\min} . Then after each subsequent iteration, Δ_c was computed. If $\Delta_c > 0$, ζ was set ζ_{\max} and if $\Delta_c < 0$, ζ was set ζ_{\min} .

Hyperparameter selection

When choosing the thresholds for ζ , the goal is to have ζ_{\max} as high as possible without skipping over regions where a keyword is located. This is based on the intuition gained from Equation 3.2. To locate this point, the AUC and EER development set scores were plotted for increasing values of ζ_{\max} which could then be used to determine the optimal value for ζ_{\max} . This is shown in Figure 4.7.

The plot shows that there appears to be a large amount of variance in the metrics as ζ_{\max} is increased. For most values, good AUC scores were accompanied by bad EER values and visa versa. It is proposed that is due to certain values of ζ_{\max} culminated in windows of overlap that aligned with the keyword contained in the utterance better, purely by chance and subsequently improving the score. However, for $\zeta_{\max} > 17$ there was a significant drop off in performance, as both metrics began to worsen consistently. Therefore, $\zeta_{\max} = 10$ was chosen as it was large enough to have a significant impact on runtime speed-up, whilst having an improved AUC score and an EER close to the baseline. To ensure Grad DTW captures enough information in the decreasing regions, $\zeta_{\min} = 3$ was chosen to keep in line with the current basine implementation.

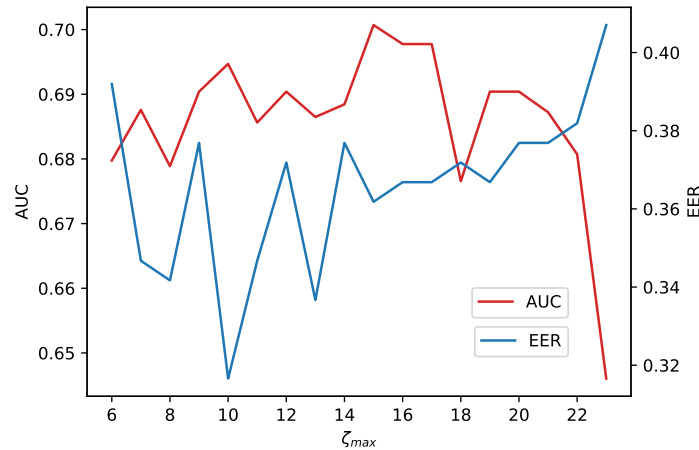


Figure 4.7: AUC and EER on the development set for increasing values of ζ_{\max} for the Grad DTW optimisation.

4.6.6. GRAM DTW

To update ζ after each iteration, Equation 3.3, which describes the momentum relation, was implemented as function. In each first iteration of the `while` loop in Algorithm 4.3, ζ was initialised to ζ_{\min} . Then, after each subsequent iteration, the current value for ζ as well as Δ_c was parsed to the momentum function which then returned an updated ζ .

Hyperparameter selection

Whilst the previous optimisations required only a single hyperparameter to be tuned, GRAM DTW has four parameters: ζ_{\min} , ζ_{\max} , k , and β . This makes selecting the optimal values for each parameter significantly more complicated. To start, the gradients Δ_c needed to be scaled by an appropriate value such that they would influence ζ . To determine what this value should be, Δ_c was observed for a few iterations of the keyword spotter on the development set. Typically, these values were in the order of between 10^{-3} and 10^{-5} . An intuitive value of $k = 10^5$ was chosen which scales Δ_c to roughly the same order of magnitude of ζ . Whilst large values for Δ_c might result in a term which is greater than 10^2 , this should still be rectified by the β term in Equation 3.3.

The parameter $\zeta_{\min} = 3$ was fixed to be inline with the Grad DTW experiments. This leaves ζ_{\max} and β to be determined. We started by setting the lower bound of $\zeta_{\max} = 6$. A general rule when using momentum in machine learning is to set $\beta = 0.9$. So, to determine a good value for ζ_{\max} , β was set to 0.9 and ζ_{\max} was incrementally increased until the AUC and EER scores on the development set worsened consistently, as shown in Figure 4.8.

Whilst this plot resembles the one generated for Grad DTW, it is plausible that this is due to a non-ideal value for β . Despite this, there were regions where large values for ζ_{\max} corresponded to comparable results to the baseline. From this plot $\zeta_{\max} = 19$ was determined as it was the value for ζ_{\max} before both metrics trended to consistently be worse. Next, the same process was repeated for β , where β was varied from 0.85 – 0.98. From these results $\beta = 0.925$ was chosen.

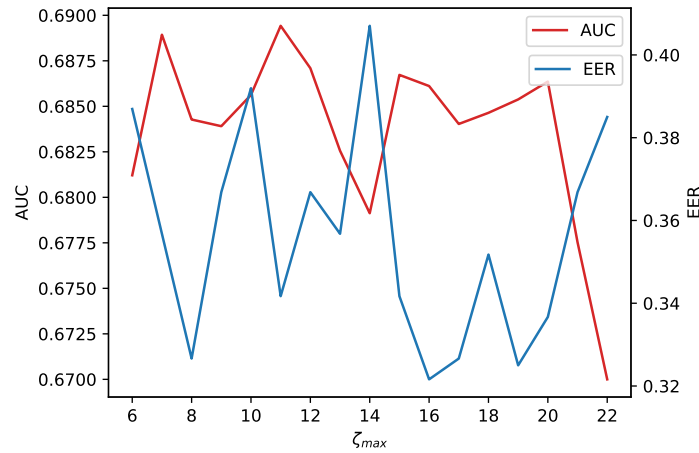


Figure 4.8: AUC and EER for increasing values of ζ_{\max} for the GRAM DTW optimisation.

4.6.7. Summary

The final values for the hyperparameters of each optimisation are summarised in Table 4.3.

Table 4.3: Summary of the final values of all hyperparameters for each optimisation.

Algorithm	Hyperparameter	Value
S-C band DTW	r	0.1
Itakura band DTW	m_1	1.35
P^* DTW	r	0.125
Grad DTW	ζ_{\min}	3
	ζ_{\max}	10
GRAM DTW	ζ_{\min}	3
	ζ_{\max}	19
	k	10^5
	β	0.925

4.7. Chapter summary

In this chapter we detailed our evaluation metrics, data sets and system architecture. We briefly outlined how each optimisation was implemented and then presented a detailed description on each optimisations hyperparamter selection. This was accomplished by varying each hyperparameter, and observing the resulting AUC and EER scores on the development set. From this, and by taking the respective speed increases into account, appropriate values for each respective hyperparameter were selected. In the next chapter we evaluate each optimisation on the unseen test set, and present our findings.

Chapter 5

Experimental results and discussion

In this chapter, each optimisation is evaluated on the aforementioned unseen test set and the results are presented. First, each optimisation is compared to Classic DTW individually, then these results are summarised and used to compare the performance of all the optimisations against each other.

5.1. Results

Using the hyperparameters detailed in Section 4.6.7, the individual results of each optimisation are presented below and subsequently summarised at the end of this section. As described in Section 4.2, the test set consisted of 480 utterances, totalling one hours worth of speech. Only the keyword *government* was searched for, for which 12 individual recordings of the keyword were used as templates. Whilst metric scores on the development set are presented, they are strictly not used for comparison. Values for each metric (AUC and EER) are expressed as percentages (%), i.e. an AUC of 100% represents an area under the curve of 1.

5.1.1. S-C band

The final results for the S-C band optimisation are presented in Table 5.1. The AUC score was marginally better (by 0.1%) than the classical algorithm, whilst both algorithms scored equally on EER. A factor of 2.7 speed-up was observed, with no loss in performance.

Table 5.1: Development and test set results for the S-C band optimisation. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	Speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
SC-band DTW	68.10	60.12	34.67	42.21	4.5	2.7

5.1.2. Itakura band

The final results of the Itakura band optimisation's are tabulated in Table 5.2. The AUC score, like the S-C band, was marginally better (0.1%) than the classical algorithm, whilst the EER was 2.3% worse than the baseline. The optimisation resulted in a speed-up factor of 3.2.

Table 5.2: Development and test set results for the Itakura band optimisation. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
Itakura-band DTW	68.66	60.12	35.18	43.22	4	3.2

5.1.3. P^* DTW

The results for the P^* optimisation are presented in Table 5.3. The optimisation performance was slightly worse than Classic DTW, with the deterioration in AUC and EER scores relative to the baseline being 0.5% and 4% respectively. For this degradation in performance, only a 1.1 speed-up factor was observed.

Table 5.3: Development and test set results for the P^* DTW optimisation. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
P^* DTW	65.80	59.78	36.68	43.97	11.5	1.1

5.1.4. Cached D DTW

Cached D DTW's final results are tabulated in Table 5.4. As expected, since cached D DTW computes the DTW exactly, and uses the classical keyword spotters value for ζ , the performance scores were identical. The optimisation did however result in a speed-up factor of 2.6 while remaining an exact implementation of DTW.

Table 5.4: Development and test set results for cached D DTW. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
Cached D DTW	68.13	60.06	34.67	42.21	4.5	2.6

5.1.5. Grad DTW

The results for Grad DTW can be found in Table 5.5. There was minor degradation in the AUC score (1%). However, a 5% improvement in EER was observed. Grad DTW resulted in a speed-up factor of 2.

In a further experiments, the best performing global constraint region optimisation, the Itakura band, was combined with Grad DTW. This resulted in a degradation in both AUC (1%) and EER (3.5%). However, the resulting speed-up factor was 5.2.

Table 5.5: Development and test set results for Grad DTW. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
Grad DTW	68.48	59.54	38.59	40.20	6	2
Itakura-Grad DTW	67.39	59.48	37.69	43.72	2.5	5.2

5.1.6. GRAM DTW

The GRAM DTW results on the test set are presented in Table 5.5. AUC was marginally lower than the baseline (0.3%) whilst the EER worsened by 3%. Despite this, a speed-up factor of 3.6 was obtained.

In further experiments, the best performing global constraint region optimisation, the Itakura band, was combined with GRAM DTW. This resulted in a slight degradation in AUC (1%) despite the EER improving by 3%. The resulting speed-up factor was 12.5.

Table 5.6: Development and test set results for GRAM DTW. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
GRAM DTW	68.13	59.83	33.67	43.47	3.5	3.6
Itakura-GRAM DTW	68.10	59.45	34.67	41.46	1	12.5

5.2. Summary of results

The aforementioned results are summarised in Table 5.7. The best scores for each metric are highlighted in bold. A composition of all the ROC plots is displayed in Figure 5.1. Both the S-C band and the Itakura band achieved the highest AUC score (60.12) whilst Grad DTW produced the lowest EER (40.20). The combination of GRAM DTW and

the Itakura band resulting in the largest speed-up factor (12.5), whilst the P^* DTW optimisation yielded the lowest speed-up factor (1.1).

Table 5.7: Development and test set results. The best scores for each metric are shown in bold. Runtime and speed-up are with reference to the test set only.

Algorithm	AUC		EER		Time (min)	speed-up (factor)
	Dev	Test	Dev	Test		
Classic DTW	68.13	60.06	34.67	42.21	12.5	1
SC-band DTW	68.10	60.12	34.67	42.21	4.5	2.7
Itakura-band DTW	68.66	60.12	35.18	43.22	4	3.2
P^* DTW	65.80	59.78	36.68	43.97	11.5	1.1
Cached C DTW	68.13	60.06	34.67	42.21	4.5	2.6
Grad DTW	68.48	59.54	38.59	40.20	6	2
GRAM DTW	68.13	59.83	33.67	43.47	3.5	3.6
Itakura-Grad DTW	67.39	59.48	37.69	43.72	2.5	5.2
Itakura-GRAM DTW	68.10	59.45	34.67	41.46	1	12.5

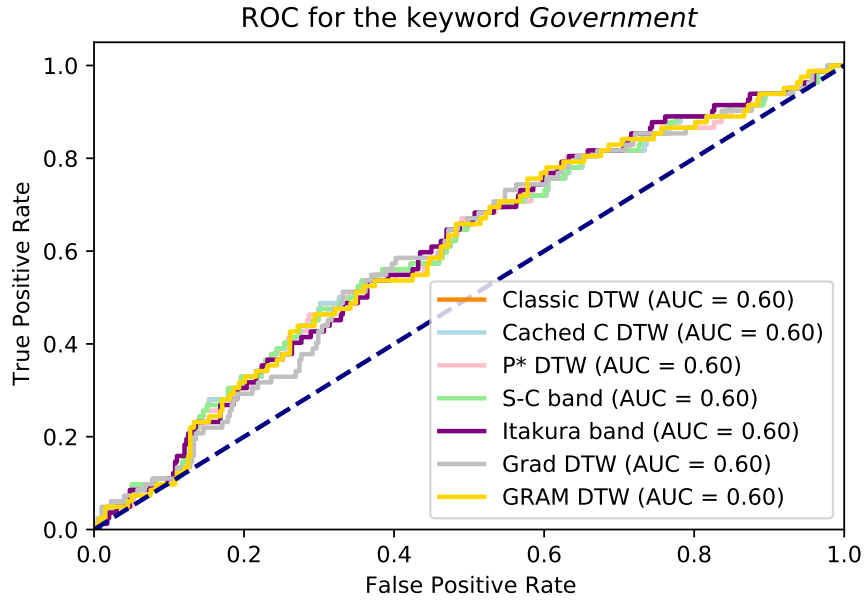


Figure 5.1: A composition of ROC plots for the results generated by Classic DTW and all the implemented optimisations.

5.3. Discussion

In this section, we discuss some general observations and compare the performance of each optimisation.

5.3.1. Observations

As indicated by the ROC plots in Figure 5.1, all optimisations achieve either identical or very similar performance to the classical DTW implementation. This is further emphasised by the results presented in Table 5.7 where it is evident that all optimisations had AUC and EER scores within 1% and 3.5% respectively of the classical DTW baseline.

Whilst it is reiterated that the results obtained on the development set are strictly not used for comparison, it is clear that the scores obtained on this data are substantially better. It is believed that this is due to the small size of this data set, where a few *easy* utterances might influence the scores by a large margin. It should be noted that the scores obtained on the test set for classical DTW are within 5% of those presented in [11] for the same keyword, and where 4 times as many templates were used.

5.3.2. Performance

Interestingly, classic DTW did not achieve the highest AUC score. Rather, it was shared between the S-C band and Itakura band optimisations. Both make use of a global constraint region, limiting the number of elements considered when computing γ . As eluded to in Sections 4.6.1 and 4.6.2, this marginal performance increase could be a result of warping paths that correspond to an incorrect, yet low DTW cost alignment being excluded by the relatively small global constraint regions of each respective optimisation imposed, thus improving the metric scores marginally.

Whilst the objective of P^* DTW was to implement a linear runtime optimisation, due to the large value for r required to achieve acceptable performance as described in Section 4.6.3, this was not achieved. P^* DTW also resulted in the worst EER score of 43.97.

The other three self formulated optimisations proposed as part of this work performed very well compared to those found in the literature. Cached D DTW had the same performance as classic DTW whilst resulting in a speed-up factor of 2.6. Since this particular optimisation significantly reduces the number distances computed for the distance matrix, an even larger speed-up was expected. The fact that this was not achieved emphasises how much of DTW's computational time is spent the nested `for` loops in Algorithm 2.1.

GRAM DTW had a marginally better AUC score than Grad DTW but a worse EER. Despite this, GRAM DTW's performance is within 3% of the baseline whilst being 40% faster than Grad DTW. This was achieved by the increased maximum frame skip threshold (ζ_{\max}) that the addition of momentum allowed. By combining the Itakura band optimisation with both Grad DTW and GRAM DTW, a substantial further speed increase was observed. Most notably, for a small performance degradation in AUC ($< 2\%$) and a slight improvement in EER, a speed-up of more than an order of magnitude was observed. The 12.5 speed-up factor was more than double the next best which was produced by combining Grad DTW and the Itakura band and almost 3.5 times faster than GRAM DTW by itself whilst having an improved EER score.

5.4. Chapter summary

In this chapter we presented the results for all implemented optimisations. These include: the S-C band, Itakura band, Cached D DTW, P^* DTW, Grad DTW, and GRAM DTW. All evaluations were performed on the test set described in Section 4.2.2. In this chapter we also discussed and interpreted our results. Of these optimisations, all had AUC and EER scores within 1% and 3.5% respectively of the baseline DTW keyword spotter. The results obtained from these experiments showed that for little to no impact on performance, runtime speed-up can be achieved. Most notably, by combining the Itakura band and GRAM DTW, a speed-up factor of 12.5 was achieved, with minimal performance impact.

Chapter 6

Summary and Conclusion

DTW is a major bottleneck in keyword spotting due to its computational complexity. This report investigated existing DTW optimisation techniques to reduce runtime in the literature, and formulated and investigated four new DTW optimisation methods that utilise the keyword spotting architecture in question.

We began by introducing the fundamental concepts of DTW and its applications, with particular emphasis notably keyword spotting which is the particular application to which DTW is currently applied. Existing optimisation techniques in the literature were reviewed, and two of these were implemented, namely: the Sakoe-Chiba band and the Itakura band. We then proposed four new DTW keyword spotter optimisations, which utilised both the inherent qualities of DTW and the specific keyword spotting architecture to which they are applied. These included introducing unique global constraint regions based on the DTW warping paths of previous windows of overlap, cost matrix caching, and the introduction of a dynamic frame skip. These optimisations were named P^* DTW, cached D DTW, Grad DTW and GRAM DTW respectively, the latter introducing momentum to Grad DTW.

All optimisations were evaluated experimentally by implementing our own keyword spotter and their performance was assessed using the metrics AUC, EER, and the achieved speed-up factor relative to classical DTW. In addition, ROC curves were utilised as a means to better visualise performance. Separate development and test sets were defined using utterances from a corpus of South African broadcast news (SABN). Whilst keyword spotting in practice searches for many keywords, to simplify our analysis and due to computational constraints, only one keyword was chosen. Of these optimisations, all had AUC and EER scores within 1% and 3.5% respectively of the baseline DTW keyword spotter. One of the proposed optimisations, a combination of the Itakura band and GRAM DTW, achieved the highest speed-up factor of 12.5, which is more than an order of magnitude faster than classic DTW. This speed increase came at the cost of a small (1%) degradation in AUC but resulted in whilst a 1.8% improvement in EER. In addition, optimisations were presented that had no impact on the keyword spotter's performance at all (S-C band and cached D DTW), whilst still achieving a reduction in computation time.

6.1. Future work

Due to limited computing power, the test set size was limited, and we could only search for one keyword. It is suggested that in future work, each optimisation should be tested on larger test set, and for multiple keywords. This will provide substantially more robust results, which can then be used to infer if any of the optimisations should be implemented in a real-world keyword spotter. These results should also be verified for the current CNN keyword spotter, which is trained on the DTW scores.

Bibliography

- [1] S. Salvador and P. Chan, “FastDTW: Toward accurate dynamic time warping in linear time and space,” *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.
- [2] L. Salhi and A. Cherif, “Robustness of auditory teager energy cepstrum coefficients for classification of pathological and normal voices in noisy environments,” *The Scientific World Journal*, vol. 2013, 05 2013.
- [3] H. Sakoe and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.
- [4] L. Lerato and T. Niesler, “Feature trajectory dynamic time warping for clustering of speech segments,” *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 6, no. 1, 2019.
- [5] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, “The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” *Data Mining and Knowledge Discovery*, vol. 31, pp. 606–660, 2017.
- [6] D. J. Bemdt and J. Clifford, “Using dynamic time warping to find patterns in time series,” New York University, 44 West 4th Street, New York, New York 10012-1126, Tech. Rep., 4 1994.
- [7] R. Varatharajan, G. Manogaran, M. K. Priyan, and R. Sundarasekar, “Wearable sensor devices for early detection of alzheimer disease using dynamic time warping algorithm,” *Cluster Computing*, vol. 21, no. 1, pp. 681–690, 2018.
- [8] S. Sempena, N. U. Maulidevi, and P. R. Aryan, “Human action recognition using dynamic time warping,” in *ICEEI 2011*. IEEE, 2011, pp. 1–5.
- [9] M. G. M. Gasser, A. Arzt, and G. Widmer, “Automatic alignment of music performances with structural differences,” in *ISMIR 2013*, 2013.
- [10] J. J. Deng and C. H. Leung, “Dynamic time warping for music retrieval using time series modelling of musical emotions,” *IEEE Transactions on Affective Computing*, vol. 6, no. 2, pp. 137–151, 2015.
- [11] R. Menon, H. Kamper, J. Quinn, and T. Niesler, “Fast ASR-free and almost zero-resource keyword spotting using DTW and CNNs for humanitarian monitoring,” in *Interspeech 2018*. ISCA, 09 2018, pp. 2608–2612.
- [12] F. Itakura, “Minimum prediction residual principle applied to speech recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

- [13] S. Srikanthan, A. Kumar, and R. Gupta, “Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery,” in *ICCCT 2011*. IEEE, 2011, pp. 394–398.
- [14] B. C. Giao and D. T. Anh, “Similarity search in multiple high speed time series streams under dynamic time warping,” in *NICS 2015*. IEEE, 2015, pp. 82–87.
- [15] E. Dijkstra and C. Piguet, “On minimizing memory in systolic arrays for the dynamic time warping algorithm,” *Integration*, vol. 4, no. 2, pp. 155–173, 1986.
- [16] C. J. Tralie and E. Dempsey, “Exact, parallelizable dynamic time warping alignment with linear memory,” *arXiv preprint arXiv:2008.02734*, 2020.
- [17] R. Hibare and A. Vibhute, “Feature extraction techniques in speech processing: A survey,” *International Journal of Computer Applications*, vol. 107, no. 5, 2014.
- [18] M. J. Alam, P. Kenny, and D. O’Shaughnessy, “Robust feature extraction for speech recognition by enhancing auditory spectrum,” in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.
- [19] M. Hasan, M. Jamil, G. Rabbani, and M. S. Rahman, “Speaker identification using mel frequency cepstral coefficients,” *ICECE*, vol. 3, pp. 565 – 568, 12 2004.
- [20] Q. Li, Y. Yang, T. Lan, H. Zhu, Q. Wei, F. Qiao, X. Liu, and H. Yang, “MSP-MFCC: Energy-efficient MFCC feature extraction method with mixed-signal processing architecture for wearable speech recognition applications,” *IEEE Access*, vol. 20, pp. 1–9, 03 2020.
- [21] L. Muda, M. Begam, and I. Elamvazuthi, “Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques,” *Journal of Computing*, vol. 2, pp. 138 – 143, 03 2010.
- [22] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.
- [23] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.
- [24] H. Kamper, F. Wet, T. Hain, and T. Niesler, “Resource development and experiments in automatic South African broadcast news transcription,” in *SLTU 2012*, 06 2012.
- [25] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A llvm-based python jit compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

Appendix A

Project Planning Schedule

Table A.1: Project goals and planned time frame.

Time Frame	Planned Goals
2020/07/27 – 2020/08/09	Research DTW and related work, as well as common optimisations.
2020/08/10 – 2020/08/16	Experiment with classic DTW.
2020/08/17 – 2020/08/23	Implement the keyword spotter, and get baseline performance for classic DTW.
2020/08/23 – 2020/09/06	Implement pre-existing optimisations and determine which ones are worth further investigation.
2020/09/14 – 2020/10/04	synthesise and implement custom optimisations unique to the DTW keyword spotter architecture.
2020/10/05 – 2020/10/07	Tune hyperparameters and run all implemented optimisations on the Test set.
2020/10/08 – 2020/11/01	Complete report and submit.

Appendix B

Outcomes Compliance

Table B.1: ECSA outcomes and how they are achieved.

ECSA Outcome	Description of Outcome in Report	Chapter(s)
ELO 1. Problem Solving	The high level problem statement requested DTW optimisations to be investigated for keyword spotting. This required the problem to be broken up into smaller sub-problems: Understanding and implementing DTW, Understanding and implementing keyword spotting, understanding and implementing pre-existing optimisations, synthesising and implementing our own new optimisations. The problem itself broad, somewhat ill-posed and the solution open-ended.	Chapters 1 through 6
ELO 2. Application of scientific and engineering knowledge	This project required extensive engineering knowledge on data science, mathematics and signal processing. Not only was this knowledge used to understand problems, but it was also required to synthesise and implement solutions	Chapter 2 through 4
ELO 3. Engineering Design	The engineering design process was followed when developing and implementing complex solutions. This entailed designing a system, implementing it and then evaluating its performance. This was then used to improve the system and repeat the process until results were satisfactory.	Chapters 2 through 5

Table B.2: ECSA outcomes and how they are achieved. (continued)

ECSA Outcome	Description of Outcome in Report	Chapter(s)
ELO 4. Investigations, experiments and data analysis	This project required extensive research as most of the content was not learnt in previous engineering courses. The project required the investigation of pre-existing optimisations, as well as formulating new ones. Experiments were then conducted to illustrate the performance of these optimisations. The results from these experiments were then presented, interpreted and analysed to evaluate the results.	Chapters 2 through 6
ELO 5. Engineering methods, skills and tools, including Information Technology	This project is software based, and required extensive computation and simulation to evaluate the performance of each programmed solution. This was accomplished in the Python programming environment. This project also required extensive data handling, where it was required to create our own development and test sets to evaluate the performance of our solutions.	Chapters 4 and 5
ELO 6. Professional and technical communication	This technical report follows appropriate structure, style and language. The report was developed using \LaTeX	Entire report
ELO 8. Individual work	All work was conducted by myself.	Entire report
ELO 9. Independent Learning Ability	This project required research on DTW and its applications, pre-existing DTW optimisations, keyword spotting, ASR, acoustic feature vectors, evaluation metrics and more.	Chapters 2 and 4