MONASH University

Time Series Classification at Scale

Chang Wei Tan

Doctor of Philosophy

A thesis submitted for the degree of *Doctor of Philosophy* at

Monash University in *2018*

Clayton School of Information Technology

Copyright notice

Abstract

Time series data are growing at an exponential pace. This growth is observed in applications such as the latest Earth observation missions that produce trillions of time series data, generated from high resolution satellite images taken from the European Space Agency's Sentinel-2 satellites. Each pixel in the image is a time series describing the evolution of the area it represents. The demand of using data in this scale is critical for producing more accurate classification. However, current research in time series classification (TSC) lags behind this demand of data. This is simply because most research in TSC was done with datasets on the scale of thousands. Adding on to this, the most accurate TSC algorithm, Hierarchical Vote Collection of Transformation-based Ensembles (HIVE-COTE) is impractical for large real-world applications because it consists of ensembles-based classifiers with high complexity, that requires huge training and classification time. The objective of this Ph.D. is to develop scalable TSC algorithms that can handle time series datasets of this scale. More specifically, this research focuses on the Nearest Neighbour (NN) classifiers that form the Ensembles of Elastic Distances (EE), a core component of HIVE-COTE.

A novel algorithm, Time Series Indexing (TSI) is proposed to reduce the classification time for the NN classifier coupled with Dynamic Time Warping distance (NN-DTW). TSI indexes time series data in DTW-induced space, which was not possible before. Classification performed with TSI is 1,000 times faster than the NN-DTW algorithm. Fast Warping Window Search (FASTWWS) and Fast Ensembles of Elastic Distances (FASTEE) are proposed to reduce the training time of EE by two orders of magnitude. This reduction is achieved by learning the parameters efficiently and introducing new lower bounds for distance measures used in EE that do not have a lower bound. The tighter Enhanced lower bound is proposed for DTW, as the majority of the DTW lower bounds are sometimes not effective in pruning candidates at larger warping windows, as shown empirically. Combining all these algorithms will reduce the complexity of EE, which in turn will make HIVE-COTE more practical.

Furthermore, it is important to apply TSC to real-world problems such as improving railway track maintenance. In collaboration with the Institute of Railway Technology (IRT) at Monash University, a study was conducted to investigate the practicality of TSC in improving railway track maintenance. IRT has more than 10 years of railway track data and provides a good platform to demonstrate the application of TSC in real-world problems. Improving and optimising railway track maintenance is crucial in reducing operation and maintenance costs. The study shows that k-NN-DTW is able to predict maintenance (tamping) effectiveness with high accuracy.

Publications During Enrolment

Publications included in this thesis

1. C. W. Tan, G. I. Webb, and F. Petitjean, “Indexing and classifying gigabytes of time series under time warping,” in *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 282–290, SIAM, 2017
2. C. W. Tan, G. I. Webb, F. Petitjean, and P. Reichl, “Tamping effectiveness prediction using supervised machine learning techniques,” in *Proceedings of the First International Conference on Rail Transportation (ICRT)*, 2017
3. C. W. Tan, M. Herrmann, G. Forestier, G. I. Webb, and F. Petitjean, “Efficient search of the best warping window for dynamic time warping,” in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 225–233, SIAM, 2018 - **Best Research Paper Award**

Other publication not included in this thesis

4. C. W. Tan, P. Reichl, G. I. Webb, and F. Petitjean, “Machine learning approaches for tamping effectiveness predictions,” in *Proceedings of the 2017 International Heavy Haul Association Conference (IHHA2017)*, IHHA, 2017

Thesis Including Published Works Declaration

In accordance with Monash University Doctorate Regulation 17.2 Doctor of Philosophy and Research Master's regulations the following declarations are made:

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 3 original papers published in top data mining and railway engineering conferences and 2 submitted publications. The core theme of the thesis is 'Scalable time series classification'. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Clayton School of Information Technology and Institute of Railway Technology under the supervision of Professor Geoffrey Webb, Dr François Petitjean and Dr Paul Reichl.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapters 3, 4, 5, 6 and 7, my contribution to the work involved the following:

Thesis Chapter	Publication Title	Status (*published, in press, accepted or returned for revision, submitted*)	Nature and % of student contribution	Co-author name(s) Nature and % of Co-author's contribution*	Co-author(s), Monash student Y/N*
3	Indexing and classifying gigabytes of time series under time warping	*Published*	75%. Concept, conducting experiments and writing the manuscript.	1. François Petitjean, Concept and input into manuscript. 15% 2. Geoffrey Webb, Supervised study and input into manuscript. 10%	1. No 2. No
4	Efficient search of the best warping window for dynamic time warping	*Published*	65% Concept, conducting experiments and writing the manuscript.	1. Matthieu Herrmann, Conducting experiments and input into manuscript. 10% 2. Germain Forestier, Concept and input into manuscript. 5% 3. François Petitjean, Concept, conducting experiments, and writing into manuscript. 15% 4. Geoffrey Webb, Supervised study and input into manuscript. 5%	1. No 2. No 3. No 4. No
5	FastEE: Fast Ensembles of Elastic Distances for time series classification	*Submitted*	80% Concept, conducting experiments and writing the manuscript.	1. François Petitjean, Supervised study and input into manuscript. 10% 2. Geoffrey Webb, Supervised study and input into manuscript. 10%	1. No 2. No
6	Elastic bands across the path: A new framework and methods to lower bound DTW	*Accepted*	80% Concept, conducting experiments and writing the manuscript.	1. François Petitjean, Supervised study and input into manuscript. 10% 2. Geoffrey Webb, Concept and input into manuscript. 10%	1. No 2. No

7	Tamping effectiveness prediction using supervised machine learning techniques	*Published*	70%. Concept, designing the system, conducting experiments and writing the manuscript.	1. Paul Reichl, Domain knowledge and input into manuscript. 10% 2. François Petitjean, Supervised study and input into manuscript. 10% 3. Geoffrey Webb, Supervised study and input into manuscript. 10%	1. No 2. No 3. No

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student signature: [redacted] **Date: 27 March 2019**

The undersigned hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor signature: [redacted] **Date:** 28 March 2019

Acknowledgements

First and foremost, I would like to express my gratitude to all my supervisors, Professor Geoff Webb, Dr François Petitjean and Dr Paul Reichl for their continuous support and guidance throughout this whole Ph.D. period. They have been very encouraging and motivating. I am truly grateful for their invaluable advice on both research and my career path, which had and will continue to help me grow as a researcher. As I have always told people I met during my Ph.D., I feel extremely lucky to have all three of them as my supervisors. From a fresh Engineering graduate who knows nothing about machine learning and data science, to now developing various competitive algorithms to solve real-world problems. All these would have been impossible without them.

I am also very grateful to my collaborators especially Professor Germain Forestier and Dr Matthieu Herrmann for the hard work that they put in, and the constructive comments that facilitated and improved my Ph.D. publications.

I would like to also thank my panel members Professor Bala Srinivasan, Professor Bart Goethals (University of Antwerp), A/Prof Michael Brand, A/Prof Vincent Lee, Dr Ron Steinfield, Dr Christoph Rudiger, Dr Peter Tischer, and Dr Lan Du for all the insightful discussion and suggestion that greatly improves the work in my Ph.D.. Their suggestion and feedback have given me lots of useful directions for my Ph.D. research.

I want to also acknowledge the support from the Institute of Railway Technology (IRT) at Monash University. Mr Ravi Ravitharan, Mr Glenn Hardie, Mr Cameron Thompson, and Mr Joshua White had equipped me with the required domain knowledge on railway maintenance and providing the required data for my Ph.D. work.

Furthermore, I greatly appreciate the various workshops, seminars and networking events organised by the faculty that help improving my research and communication skills, special thanks to Professor Sue McKemmish, Ms Hellen Cridland, Ms Danette Deriane and Ms Julie Holden. I also appreciate the hard work and valuable time spent by Ms Danette Deriane for organising my milestones seminars as well as her assistance throughout my Ph.D. journey; Ms Julie Holden for her help and support for organising my dissertation.

More importantly, I am very grateful for the financial support provided by my supervisors, the university, and the faculty. I would like to take this opportunity to thank the Society of Industrial and Applied Mathematics (SIAM), for awarding me with the SIAM Student Travel Award, which allows me to attend the SIAM Data Mining conference for two consecutive years and gaining invaluable experience. This research was also supported by an Australian Government Research Training Program (RTP) Scholarship.

I would like to also thank Dr Ying Li, Dr Manai Giuseppe, and Dr Emin Aksehirli, who are my internship mentors at DataSpark Singapore, for the insprising discussion and guidance throughout my internship with them. They have brightened my future career as a data scientist and software developer.

Finally, I want to express my deepest gratitude and appreciation to my family, parents, siblings, and my partner, Katrina for their unconditional love, encouragement and support that not only helped me go through my Ph.D. journey but also for everything that I have done throughout my life.

Contents

List of Figures

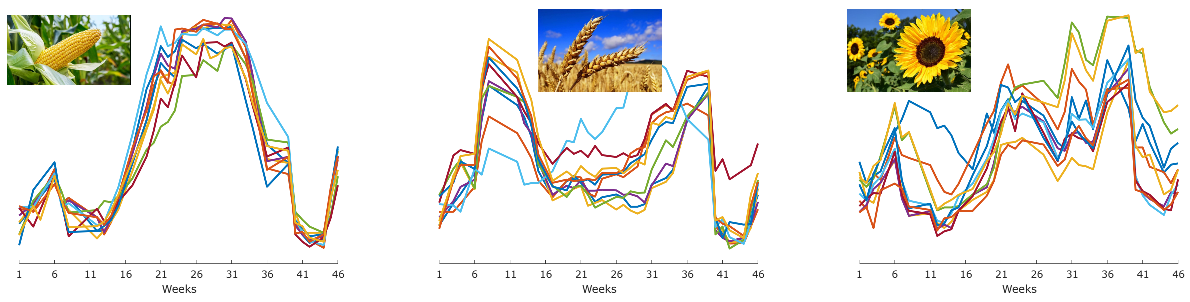
List of Tables

Chapter 1

Introduction

In 2017, the world's biggest retailer – Walmart generated 2.5 petabytes of data per hour [5]. That is equivalent to 1 million customer transactions every hour [5]. Walmart's Senior Statistical Analyst Naveen Peddamail said that *"If you can't get insights until you've analysed your sales for a week or a month, then you've lost sales within that time. If you can cut down that time from two or three weeks to 20 or 30 minutes, then that saves a lot of money for Walmart and stopped us losing sale."* [5]. This indicates three things: (1) petabytes of data are now being produced every hour, (2) analysing data of this scale is extremely important to create finer models that enable complex business decisions and (3) this large amount of data needs to be processed in a timely manner.

The demand to learn from large amounts of data is observed for time series data as well. A time series is a collection of data ordered in time that measures the evolution (dynamics) of a phenomenon represented by the data. Time series classification (TSC) is a tool to learn from time series data. It labels an unknown time series from a set of predefined labels. For example, labelling time series created from satellite images (describe in more detail later) as wheat, maize or sunflower fields. This is illustrated in Figure 1.1. Training TSC algorithms using this wealth of data is important to create finer and more accurate models. Unfortunately, existing TSC algorithms do not scale beyond datasets with more than 10 thousand time series instances. The reason is simply because they were tested and designed with the benchmark time series archive that holds 85 datasets, the largest of which contains less than 10 thousand time series instances [6].



(a) Maize (b) Wheat (c) Sunflower

Figure 1.1: Examples of satellite image time series (SITS). Different colours represent different time series

As a beacon for this research, TSC has been demonstrated on two real-world applications. In particular, TSC has been applied on Earth observation missions and railway track maintenance. The latest Earth-monitoring satellite, Sentinel 2 launched by the European Space Agency (ESA), is providing a full picture of Earth every five days at 10-60m resolution [7]. By assimilating the Earth's geographic areas into pixels, these images are transformed into time series data, which are used to create temporal land-cover maps that describe the evolution of that geographic area over time [8, 9, 10]. Each new satellite image, taken every five days, gives a new observation for the areas in the image. Earth has a surface area of 500 million km^2 and considering a 10m$\times$10m area per pixel as a time series, this translates to more than a trillion time series being produced [10]. The ability to analyse these high-resolution satellite image time series (SITS) will have a significant impact in many domains especially in the agriculture industry and for environmental monitoring.

Moving back to Earth, the Institute of Railway Technology (IRT) at Monash University has been assisting for decades mining companies in monitoring heavy haul tracks using Instrumented Revenue Vehicles (IRVs) that are equipped with on-board sensors [11, 12, 13]. These sensors measure the wagon's dynamic activities to infer information about the underlying track condition [11, 12, 14]. Irregular track conditions can increase track loading, reduce infrastructure component life, and increase the risk of vehicle derailment. These issues trigger a range of track maintenance operations and renewals that are very costly. For example, one of the longest heavy haul tracks connecting ore mines in Australia spans over 1,700km long and the entire network requires maintenance. These tracks are divided into 50m sections, i.e. each of them gets a location ID. After multiple trips, over a period of time, the measurements from the IRVs at each track section forms a time series that measures the evolution of the track sections. This translates to more than 30 thousand time series generated from these mining operations. Processing and analysing data generated from these tracks will help in understanding how the track degrades and the consequences of the degradation in terms of cost and safety. With this information, railway engineers are then in a better position to estimate the timing of inspections, maintenance, and renewals, which in turn minimises the maintenance costs and improves safety.

Many of the problems in these fields are time series classification problems, where a location is labelled based on its evolution over time. The classification of time series data is different from traditional tabular data because the ordering of data attributes in time series is important in finding discriminating features. For example, differentiating a wheat, sunflower or maize field from satellite images can only be achieved if the evolution of that location can be observed over time. Figure 1.1 shows that maize has a single peak, indicating that there is only one harvesting period during the warmer months; wheat has two peaks indicating two harvesting period during winter and spring months.

The most accurate TSC algorithm to-date is the Hierarchical Vote Collective of Transformation based Ensembles (HIVE-COTE) [15]. HIVE-COTE is a meta-classifier of five state-of-the-art ensemble-based classifiers [15]. Despite the superior classification accuracy, classifying large time series datasets using these ensemble-based classifiers is very computational demanding and they need to be trained to achieve state-of-the-art performance [16]. For instance, one of the classifiers in HIVE-COTE, the Ensembles of Elastic Distances (EE) takes 2-3 weeks to train on the `ElectricDevices` dataset [6] with only 8,926 training instances and 10 years to train on a million SITS dataset. EE is an ensemble that consists of 11 Nearest Neighbour (NN) classifiers paired with various distance measures [17]. Then, to classify a query, the NN classifiers in EE need to compare the query time series to each of the instances in the training dataset. The comparison of a pair of time series with length L is computationally expensive as it typically requires $O(L^2)$ operations. Furthermore, there are four other ensembles in HIVE-COTE with similar or higher complexity than EE [15].

It is important to realise that down-sampling the data to the size that is manageable for the state-of-the-art classifiers is not sufficient to produce accurate models for these problems. The reason being, datasets on a smaller scale are not enough to capture the diversity in a dataset, which will result in creating models with high variance. Although datasets can be down-sampled through stratified sampling, where every class in the dataset will have equal amount of data, they are still not enough to represent the diversity in the dataset. For instance, there have been 30 years of research into railway track maintenance, but there is no general agreement on the best approach [18]. This is because the degradation of tracks is dependent on a combination of many complex phenomena such as soil properties and weather that are difficult to model and predict using small datasets [19, 20]. Furthermore, the creation of the land-cover maps typically requires about 100 million time series [8, 9] to be able to capture most of the surface evolution on Earth. Therefore, this thesis aims to develop scalable TSC algorithms to handle time series data at the scale of millions to billions at both training and classification time.

1.1 Motivation

The majority of TSC algorithms concentrate on similarity measures [17, 21, 22, 23, 24] that capture the temporal structure of the time series. Most similarity measures transform a time series to better match the other time series [17]. There are a dozen similarity measures that have been proposed for time series matching and will be explained in Section 2.2. These measures are normally paired with a NN classifier. When benchmarked on the UCR benchmark time series archive [6], they are not significantly different from each other in terms of classification accuracy. However, an ensemble of them (EE) is significantly more accurate than each of them [17]. The NN classifier with Dynamic Time Warping (NN-DTW) was the leading classifier [10, 16, 22, 23, 24, 25, 26] before the introduction of EE and the other ensemble-based classifiers. The research in this thesis focuses on speeding up TSC in the time domain, specifically the NN classifiers.

Current research in TSC lags behind large scale time series data. Majority of the TSC algorithms including the state of the arts are impractical for large time series datasets, as

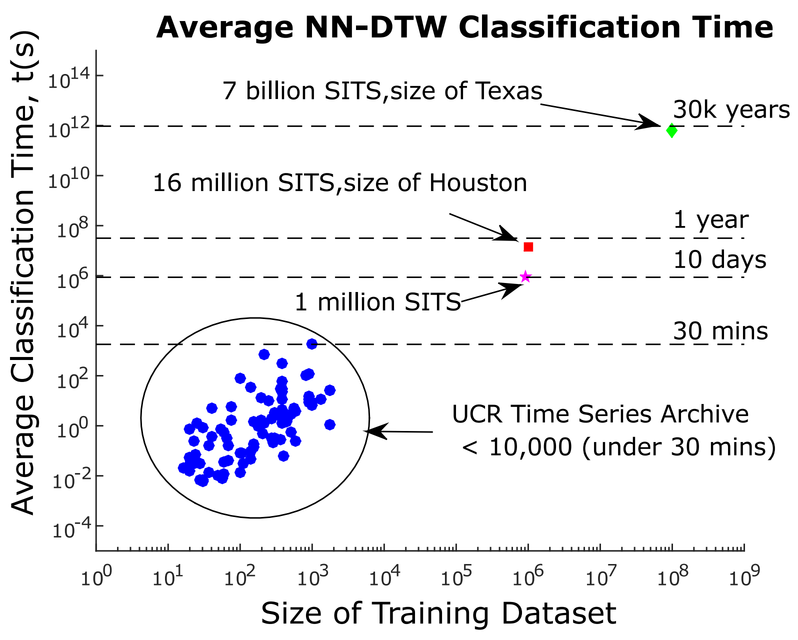


Figure 1.2: Average NN-DTW classification time on various datasets

they were designed using datasets that hold less than 10 thousand time series instances [6]. Figure 1.2 illustrates this problem. It shows the expected classification time for datasets of various sizes using NN-DTW with lower bounds. A typical classification using NN classifiers requires $O(N \cdot L^2)$ operations. This is because, for each query time series of length L, the algorithm needs to scan through all N training examples and the comparison of a pair of time series requires $O(L^2)$ operations. Lower bounds are commonly used to speed up NN classifiers by minimising the number of expensive distance computations and pruning off unpromising NN candidates [27, 28, 29, 30, 31, 32] (more details in Section 2.4). Figure 1.2 shows that classifying 16 million time series queries (a land-cover map of the size of a city) using a million training instances will take 8 to 12 months; 7 billion queries (the size of a state) using 100 million training instances is impossible; while most of the publicly available time series datasets [6] can be classified in less than 30 minutes.

Apart from lower bounds, efficient indexing strategies are also commonly used to speed up NN algorithms. It allows faster retrieval of NN at classification time. However, typical indexing structures such as the k-d tree [33] are not suitable to time series data because the ordering of data attribute in time series is important. For instance, k-d tree partitions the training dataset using the attributes of the data. And by doing so, the temporal structure of the time series, which is required for distance measures to work, is lost. Furthermore, these indexing strategy suffer from the curse of dimensionality [34], i.e. they cannot handle high-dimensional data such as time series, where it is common to have thousands of dimensions.

There are various indexing techniques developed for time series [27, 28, 35, 36, 37, 38, 39, 40, 41, 42, 43]. Most of them index time series under the Euclidean distance; some use symbolic representations such as iSAX. However, many TSC problems such as the creation of land-cover maps are better tackled with the DTW distance, as many phenomena of interest are periodic and can be modulated by weather [10]. Indexing techniques for DTW [27, 38] typically use the GEMINI framework [39] by representing the time series at a higher level representation using its lower bound function. Depending on the tightness of the lower bounds (how close the lower bound is to the distance of interest), these techniques can be ineffective at indexing the training dataset.

Most TSC algorithms including the state-of-the-art ensembles are only competitive if given the right parameter [16, 23]. This right parameter needs to be learned through cross validation from a set of values [23]. However, the learning time of most TSC algorithms including the state of the arts is prohibitive for large datasets. Consider the NN classifiers in EE, they are parametrized by one or two parameters of their paired distance measures. The best parameter value is normally learned from a set of M predefined values using leave-one-out cross validation (LOO-CV). Thus, the task can be re-framed into creating a NNs table of size $N \times M$ that gives the NN for each time series in the training set. This means that, for each instance, the algorithm has to do $N-1$ expensive $O(L^2)$ comparisons and repeat it over M values.

Furthermore, the number of values, M to learn from also significantly impacts the training time of the classifier. For instance, learning the exact best warping window (DTW's parameter and will be explained in Section 2.2) for DTW is a laborious task as it requires the enumeration of all possible warping windows, i.e. $M = L$. Hence, algorithms like EE are settled with a subset of warping windows, typically 100 windows (i.e. $M = 100$) [17, 23]. Apart from the risk of not learning the exact best warping window (parameter) and compromising on the classification accuracy, current approaches to learning the parameters are very inefficient.

Consider if the nearest neighbour for a time series T_1 with a parameter value of p is T_{25}, then the distance $d(T_1, T_{25}, p)$ will be computed at cell NN$(1, p)$. Then in order to find the nearest neighbour for T_{25}, the naïve way will compute $d(T_{25}, T_1, p)$ at cell NN$(25, p)$. However this computation is redundant because the value is the same as $d(T_1, T_{25}, p)$. Note that this example is only valid for symmetric distances, i.e. $d(S, T, p) = d(T, S, p)$ and that all the distances used satisfy this property. Moreover, some of the distance values may stay the same for a wide range of values. For instance, if $d(T_1, T_{25}, p) = d(T_1, T_{25}, p^\star)$, the naïve way will compute the same distance for p and $p^\star$, which again is redundant and inefficient. Therefore, building the table the naïve way requires $O(M \cdot N^2 \cdot L^2)$ operations, which is impractical for most real-world applications with large and long time series dataset.

Typically the best warping window for a large dataset is small [22, 44] and this is where most DTW lower bounds are effective. However, they lose their effectiveness when the window size increases. Warping window is a global constraint applied on the alignments of the two time series under the DTW distance [45]. In other words, only data-points that are within a window range can be aligned (more details in later chapters). This has the effect of speeding up NN-DTW by reducing the number of operations required in the distance computation. Efficiently computing these lower bounds at larger warping windows is important for reducing the training time of NN-DTW and EE. Recall that, learning the best warping window for DTW usually requires the enumeration of 100 windows in the range of 0 to L. This includes larger warping windows. On the other hand, tighter lower bounds [30, 31] are more expensive to compute, requiring more computations, thus not effective at speeding up NN-DTW. Moreover, most distance measures do not have a lower bound function that can be used to speed up the nearest neighbour search process.

It is also interesting and important to apply TSC to real-world problems such as improving railway track maintenance. TSC has not been applied to the challenge of improving railway track maintenance. Understanding how the track degrades is important to improve railway track maintenance. Existing track degradation models are developed uniquely for a particular track location and cannot generalise [18]. This is due to various complex phenomena such as soil properties and weather that are difficult to predict using small datasets that are unable to capture the evolution of the tracks [18]. Furthermore, historical data show that track maintenance is not always effective at restoring the track to the desired track geometry. Ineffective and unnecessary maintenance can reduce the lifetime of existing track, which is counter to the purpose of track maintenance.

To summarise, the following challenges motivate this Ph.D. research.

1. Current state-of-the-art TSC algorithms has high training and classification time complexity that prohibits them from being practical for large datasets.
2. Existing indexing techniques that speed up the NN classifier are not applicable to time series data and the DTW distance.
3. The learning algorithms for the NN classifier are inefficient because of many redundant distance computations.
4. Existing DTW lower bounds are ineffective at speeding up NN-DTW at large warping windows, which limits the potential of improving the training time.
5. Most distance measure do not have a lower bound function and thus the NN classifier that is paired with it cannot be sped up.
6. A good railway track maintenance system will minimise maintenance costs and improve safety but it is challenging to develop.

1.2 Research Questions

The main objective of this Ph.D. research is to develop scalable TSC algorithms to overcome the limitations from previous works mentioned in Section 1.1. In particular, applying NN classifiers that are paired with an distance measure to large time series datasets in the range of millions are investigated. This includes designing scalable algorithms for both training and testing time as well as optimising the distance computation. Specifically, this Ph.D. thesis aims to address the following research questions (RQ).

RQ-1: Indexing techniques speed up the NN search process but they are not applicable to time series data especially under the DTW distance. **How to index time series data under the DTW distance?**

RQ-2: Existing learning algorithms for the NN classifiers are inefficient thus not scalable. **How to improve the efficiency of learning algorithms for NN classifiers?**

RQ-3: Current DTW lower bounds are loose at large warping windows, which slows down the learning of the best warping window. **How to have tighter bounds at larger warping windows (global constraints) that are not expensive to compute?**

RQ-4: It is important to apply TSC to real-world problems such as improving railway track maintenance. **How can time series classification be applied to improve railway track maintenance?**

1.3 Contributions

This section outlines the contributions made in this Ph.D. research. This research proposes an indexing algorithm to index time series data under the DTW distance. An efficient learning algorithm for the NN-DTW classifier is proposed and extended to other distance measures with the aim of scaling up the EE classifier. New lower bounds for the major distance measures have been proposed to efficiently speed up the NN search process. A study is conducted showing that TSC can be applied to improve railway track maintenance. More formally, the following contributions (C) are made:

C-1: Chapter 3 introduces a new problem – Contract Time Series Classification to answer RQ-1. The idea is to produce the most accurate time series classifier that is constrained at classification time but without any constraints at training time. Specifically, *Time Series Indexing* (TSI) is proposed to index time series data under the DTW distance using a hierarchical K-means tree with priority search [46] and the DTW Barycenter Averaging (DBA) algorithm [47]. K-means clustering was ill-defined for DTW-induced space, but it has shown possible with DBA. The classification time of NN-DTW is reduced by representing the training set with a few average time series computed using DBA for each class [26]. TSI has been evaluated against the state-of-the-art NN-DTW algorithm on dataset with a million instances and the benchmark time series archive [6]. This research has been published as in the 2017 SIAM Data Mining Conference (SDM 2017) [1].

C-2: The work in Chapter 4 and 5 answer RQ-2. In Chapter 4, the *Fast Warping Window Search* (FASTWWS) algorithm is proposed to efficiently learn the best warping window for DTW [3]. FASTWWS minimises the number of operations required to build the NNs table by leveraging on the relationship between DTW and its warping window. FASTWWS has been evaluated against state-of-the-art algorithms on all the datasets from the benchmark time series archive [6] as well as a dataset with a million instances. This research has been published in the 2018 SIAM Data Mining Conference (SDM 2018) [3] and received the *Best Research Paper Award*.

In Chapter 5, the idea of FastWWS is extended to other distance measures with the aim of reducing the training time of the EE ensemble. The *Fast Ensembles of Elastic Distances* (FastEE) algorithm is proposed. The work also introduces new lower bounds for distance measures for which no previous bounds have been derived. FastEE has been evaluated against the standard EE classifier on the benchmark time series archive [6]. This research has been submitted to Data Mining and Knowledge Discovery Journal – the journal track for the 2019 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2019).

C-3: The third contribution of this thesis answers RQ-3. A new framework to lower bound the DTW distance, the Enhanced lower bound (LB_Enhanced), is proposed and presented in Chapter 6. LB_Enhanced exploits the tight constraints applied to the calculation of DTW. The Enhanced lower bound has been evaluated against existing DTW lower bounds on the benchmark time series archive [6]. It provides similar tightness with one of the tightest DTW lower bound while having faster computation time. The research in this chapter has been submitted to the 2019 SIAM Data Mining Conference (SDM 2019).

C-4: The study presented in Chapter 7 answers RQ-4. The aim of this study is to demonstrate the application of TSC algorithms on real-world problems such as improving railway track maintenance. This study is a collaboration with the Institute of Railway Technology (IRT) at Monash University. IRT has decades of railway track data that provide a good platform to demonstrate TSC applications. In particular, this study look at predicting the effectiveness for tamping maintenance activity using a multivariate k-NN-DTW algorithm. The system achieves 72% prediction accuracy in predicting tamping effectiveness and is able to predict the effectiveness twelve weeks before tamping. Future work includes applying scalable TSC algorithms to improve the performance, using the wealth of railway data at IRT. This study has been published to two top railway conferences – the 2017 International Conference of Rail Transportation (ICRT 2017) [2] and the 2017 International Heavy Haul Association Conference (IHHA 2017) [4].

Chapter 2

Literature Review

This chapter gives some general background and related work on time series classification (TSC) to have a high level overview of the work done in this thesis. If a more detail background or related work is required, they will be presented in the respective chapters. This chapter will also review some of the related work in scaling up time series classification. Section 2.1 gives a general overview on TSC. Section 2.2 then gives some background on distance-based classification algorithms that is used in this thesis. Then Section 2.3 reviews the related work in scalable TSC and finally Section 2.4 gives an overview on the existing lower bounds for distance measures that can be used to speed up the Nearest Neighbour (NN) classifiers. Starting with the definition of a time series given in Definition 2.0.1 where $S(i)$ denotes the i-th element of time series S.

Definition 2.0.1. *A time series S is an ordered time-variables pair of L real-valued variables, $S = \{(S(1), t(1)), (S(2), t(2)), ..., (S(L), t(L))\}$, where $t(1)$ to $t(L)$ is the timestamps for the measurements $S(1)$ to $S(L)$. The ordering of the data attribute is critical in finding the best discriminating features in time series data.*

2.1 Time Series Classification

Like any standard supervised classifiers, time series classification algorithms build a classifier from a collection of labelled time series data [17]. Definition 2.1.1 gives a formal definition of time series classification. For simplicity, the majority of work presented in this thesis assume that all time series are of the same length (as given in the publicly available benchmark datasets [6]) but can be extended to time series of different length.

Definition 2.1.1. *Time series classification is defined as, given an unlabelled time series S, assign it to a class c_i from a set $C = \{c_1, c_2, ..., c_K\}$ of predefined class, where K is the number of classes in a dataset.*

Early work on TSC involve using classifiers such as the Support Vector Machines [48] and Decision Trees [49, 50]. Some fit a model to each time series by assuming that time series in a class are generated by an underlying model or probability distribution [51]. Some of the popular ones are the Naïve Bayes [52], Hidden Markov Model [53] and Auto-Regressive models. As mentioned in [23], these models are not competitive for TSC tasks as they were proposed for other tasks, for example regression. Hence will not be discussed in this thesis.

Time series classification algorithms can be divided into 4 major categories. They are the *distance-based*, *feature-based*, *dictionary-based* and *ensemble-based* algorithms. Distance-based algorithms perform classification by comparing two time series with a similarity measure. These techniques compute a distance between two time series. They are simple, intuitive and was the leading classification algorithm before the introduction of state-of-the-art ensemble-based algorithms. This thesis focuses on TSC algorithms in this area because it is where most research in TSC are at [16, 22, 23, 24, 25, 51]. Section 2.2 will discuss this in more details. This section reviews the different types of TSC algorithms, focusing on the more accurate ones outlined in a recent benchmark on TSC algorithms [23] and the ones that contribute to the most accurate TSC algorithm – HIVE-COTE [15].

2.1.1 Feature-based Algorithms

Feature-based algorithms perform classification by learning features from the time series that best discriminate the different classes. Feature-based algorithms are more likely to benefit problems with long series as more features can be extracted from the time series [23]. Rather than using features extracted from a whole time series for classification, better results are obtained by extracting features from subsequence of the time series, also known as interval-based algorithms [23, 54].

A recent benchmark [23] ranks the Time Series Forest (TSF) algorithm [54] as one of the top nine most accurate TSC algorithms when tested on the UCR benchmark datasets [6]. TSF extracts features such as mean, standard deviation and slope from an interval of the time series. It then builds a forest of *time series trees* using the Random Forest approach, where random intervals of size $\sqrt{L}$ and $\sqrt{L}$ starting positions are selected in each node of a time series tree [54]. Then, classification is done through majority vote [54]. An extension of the TSF algorithm is the Time Series Bag of Features algorithm [55] that builds a bag of features from the same features derived from TSF. Time Series Bag of Features is also in the top nine TSC algorithms and is not significantly worse than TSF [23].

Instead of using features extracted from the subsequence, the whole subsequence can be used as the discriminating pattern between the different classes in a training dataset, also known as time series shapelets [56]. A shapelet is usually used as the splitting criterion at a node of a decision tree. However, the algorithm has very high training complexity as it needs to scan through a high number of shapelet candidates. For instance, a time series of length L has $O(L^2)$ shapelet candidates and a time series dataset of size N has $O(NL^2)$ candidates [56]. For example, the `Trace` dataset [6] with $N = 200$, $L = 275$, $\texttt{MINLEN} = 3$, and $\texttt{MAXLEN} = 275$, has 7,480,200 candidates [56]. Hence, many shapelet candidates are required to build the decision tree, which is extremely prohibitive for large datasets and long time series.

Several work has been done to improve on the original shapelet algorithm. The Fast Shapelets algorithm [57] approximates the shapelets using the Symbolic Aggregation Approximation (SAX) algorithm [58], to speed up the shapelets discovery process. SAX is a method to approximate time series using word (sequence of symbols) [58]. A SAX word is formed by first segmenting the time series into equal-width segments, where each segment is represented by its mean. This is known as Piecewise Aggregated Approximation (PAA) [41]. Then each segment is discretised using a fixed breakpoints along the y-axis and labelled with a symbol.

The Learned Shapelets algorithm [59] uses a gradient descent heuristic to speed up the shapelets discovery process. It finds k shapelets through K-means clustering and optimised using a logistic loss function for each class [59]. The Shapelet Transform algorithm [60] transforms time series data using the distance of a time series to all k shapelets, where each distance is the new data attribute. Instead of K-means clustering, it finds top k shapelets in a single scan [60]. The transformed data are used to construct the Shapelet Ensemble (SE) classifier, which is an ensemble of 8 standard classifiers [21]. SE is the second most accurate time series classifier [23] and is part of the most accurate ensemble-based classifier HIVE-COTE [15] that will be described later.

2.1.2 Dictionary-based Algorithms

Dictionary-based algorithms build a dictionary, which is a histogram that represents the observed frequency of a particular pattern or feature in the time series. Then, classification is done by comparing the histograms. These algorithms are also known as the “bag of words” algorithms, where the patterns are captured using words, a high level representation of the time series. Words are created by sliding a window over the time series, extracting all possible subsequences, and labelling each subsequence with a symbol. The Bag of Patterns (BOP) algorithm [61] builds the dictionary by labelling each subsequence with a SAX word. An extension of BOP is to combine the SAX representation with the vector-space model, forming the SAX Vector-Space Model (SAXVSM) algorithm [62]. SAXVSM builds the dictionary using the frequency of words in each class rather than the series itself. It is more scalable and accurate than the original BOP algorithm [23].

The Bag of SFA Words (BOSS) algorithm [63] is the current state of the art for dictionary-based algorithms and the third most accurate time series classifier [23]. The ensemble version of it is a core component of HIVE-COTE [15]. It builds the dictionary using SFA words (Symbolic Fourier Approximation [64]). BOSS creates the SFA words for each subsequence using truncated Discrete Fourier Transform, which makes it robust to noise [63]. It discretises each subsequence using Multiple Coefficient Binning, which finds the breakpoints by estimating the distribution of the Fourier coefficients [63].

2.1.3 Ensemble-based Algorithms

Ensemble-based algorithm is a set of classifiers, where the prediction from each classifier can be weighted to achieve better classification accuracy. A key requirement for ensemble-based algorithms to perform well, is to have diversity in the ensembles. Diversity in an ensemble can be achieved by using different classifiers; selecting different attributes for each classifier; or through different re-sampling of the training data for each classifier. There has been many growing interest in developing ensemble-based algorithms for TSC [17, 17, 21, 54, 63] in the recent years. The main reason is because ensembling improves the classification accuracy [21]. For instance, ensemble-based classifiers such as the Ensembles of Elastic Distances (EE) [17], Collective of Transformation-based Ensembles (COTE) [21] and Hierarchical Vote COTE (HIVE-COTE) [15] are all significantly more accurate than their individual components.

A few of the ensemble-based algorithms have been reviewed in the previous sections. The Time Series Forest extracts the summary statistics (features) from intervals of the time series and builds a Random Forest classifier. The BOSS ensemble builds multiple BOSS classifiers with different sliding window sizes [63]. The Shapelet Ensemble (SE) [21] is formed using 8 heterogeneous classifiers such as the k Nearest Neighbours (k-NN), Naïve Bayes, C4.5 Decision Trees [50], Support Vector Machines [48] with linear and quadratic kernels, Random Forest [65], Rotation Forest [66] and Bayesian Network [15, 21]. Instead of extracting features or words from the time series, SE transforms the time series into the shapelet representation, using the Shapelet Transform algorithm [60] (see Section 2.1.1).

The Ensembles of Elastic Distances (EE) is an ensemble that groups the distance-based classifiers [17]. It is an ensemble of nearest neighbour (NN) classifiers paired with 11 different distance measures [17]. EE is significantly more accurate that the individual NN classifiers [17] and is the fourth most accurate time series classifier without being significantly worse than the BOSS classifier [23]. More detail on these distance-based algorithms will be outlined in Section 2.2.

The Collective of Transformation-based Ensembles (COTE) is a collection of 35 classifiers in the time, shapelet, autocorrelation and power spectrum representation [21]. It contains 11 classifiers in the time domain (EE) and 8 in each of the shapelet (SE), autocorrelation and power spectrum representation. Time series data are transformed into the autocorrelation and power spectrum representation [21]. Then, the same classifiers in SE are used in these representations. Combining all the 35 classifiers in multiple domains makes COTE a more accurate classifier than the individual classifiers in their respective domains [21]. The authors [21] also demonstrated that choosing a transformation domain based on the training accuracy makes COTE worse.

COTE has its own limitations. It treats each of the classifier as a single module, meaning that the prediction from every single classifier is considered and weighted based on the training accuracy. Since there are more time domain classifiers in COTE, COTE will be more biased towards time domain classifiers [15]. The HIVE-COTE algorithm is proposed to overcome this limitation and is significantly more accurate than COTE [15]. It treats the ensemble in each domain as a module and outputs a single probabilistic prediction from each module [15]. There are five ensembles in the HIVE-COTE structure. They are the EE [17], SE [21], BOSS Ensemble [63], Time Series Forest [54] and a newly introduced Random Interval Features (RIF) Ensemble [15]. The RIF Ensemble is analogous to the autocorrelation and power spectrum representations in COTE. It uses intervals from data transformed into these spectral representations [15].

The Proximity Forest is the most recent ensemble-based classifier that is scalable and has similar classification accuracy with COTE [67]. It is also not significantly worse than ResNet – one of the deep learning model for TSC that is competitive to COTE [68]. Similar to Random Forest [65], Proximity Forest is an ensemble of proximity trees. The difference between a proximity tree and a normal decision tree is the splitting criteria of each node, where it selects a random distance measures from EE as the splitting criteria [67]. The Proximity Forest has not been compared to the work in this thesis because it has just been proposed, but nonetheless it will be an interesting and promising future work.

2.1.4 Deep Learning for Time Series Classification

Deep learning [69] has gain a lot of interest in the machine learning community in the recent years. It has been very successful in many classification tasks such as computer vision [70, 71], natural language processing (NLP) [72, 73] and speech recognition [74, 75]. These successes spark the interest in developing deep learning models for TSC [68, 76, 77]. In fact many of them such as speech recognition and NLP, share similarity with TSC tasks [68]. This thesis is not focused on deep learning methods, thus this section will only briefly review some of the major deep learning architecture for TSC for future exploration. A detailed and comprehensive review has been presented in [68].

One of the earliest work in deep learning for TSC is the work presented by [78] where a Multi-scale Convolutional Neural Network (MCNN) is proposed. MCNN is very competitive to the state-of-the-art TSC classifier COTE [78]. However, the method is complicated to deploy because it requires some heavy data preprocessing such as down-sampling and sliding windows, to prepare for the multi-scale setting. Then, Multi Layer Perceptron (MLP), Fully Convolutional Networks (FCN) and Residual Network (ResNet) are proposed for TSC in [76]. Their experiments show that FCN and ResNet are very competitive to MCNN and state of the arts such as COTE and BOSS. On the other hand, the proposed MLP is competitive with NN-DTW. More importantly, these models have small training and deploying complexity compared to MCNN and the ensemble-based classifiers. In contrast to the results in [76], a recent review of existing deep learning models [68] show that ResNet outperforms FCN when evaluated on 85 benchmark time series datasets [6] instead of the original 44 [76].

Furthermore, deep learning models have the advantage over traditional machine learning models for transfer learning [79]. The idea of transfer learning is to pre-train a model on a problem (dataset) and use the trained model to train on a new problem [79, 80]. Transfer learning is useful when the problem does not have enough labelled data [79]. A recent work [80] investigated the transfer learning approach for TSC tasks. They concluded that the choice of source dataset significantly impacts the generalisation of the models and proposed a method to predict the choice of source dataset using the DTW measure [80].

2.2 Distance-based Classification

The majority of time series classification research has been focused on similarity in the time domain. This involves comparing a pair of time series using a similarity measure and gives a distance between the two time series. Most similarity measure find the optimal alignment between the two time series in the time domain using dynamic programming. The optimal alignment is the cheapest path that goes through a $L \times L$ cost matrix. Different similarity measure have different cost of alignment. Then, the Nearest Neighbour (NN) algorithm is employed for classification. It searches for the NN within the training set and labels the query time series with the label of the NN. Algorithm 1 describes this process, which is also known as *sequential scan*. First, the distance to the nearest neighbour is initialised to infinity. Then the algorithm compares time series S to every candidate T in the training set $\mathcal{T}$. If the distance is less than the current distance to the nearest neighbour, the candidate becomes the new nearest neighbour. Note that most distance functions are parametrised by a parameter $\mathcal{P}$, which will be explained in more detail later in this section.

2.2.1 L_p-norm Distances

The L_p-norm distances are the simplest and fastest distance to compute. They have $O(L)$ complexity and provides a one-to-one matching of the time series. Among all the L_p-norms, the L_2-norm which is also known as the Euclidean distance (ED) is most commonly used to compare time series [22]. Figure 2.1 illustrates the alignment of ED between two time series

Algorithm 1: NNSEARCH($S, \mathcal{T}, \mathcal{P}$)

```
Input: $S$: Query
Input: $\mathcal{T}$: Data
Input: $\mathcal{P}$: Parameter for the distance
Result: $\mathrm{NN}$: Nearest neighbor of $S$ in $\mathcal{T}$
1 $\mathrm{NN}.\mathtt{dist} \leftarrow +\infty$
2 foreach $T \in \mathcal{T}$ do
3     if DISTANCE($S, T, \mathcal{P}$) $< \mathrm{NN}.\mathtt{dist}$ then
4         $\mathrm{NN} \leftarrow T$
5     end
6 end
7 return $\mathrm{NN}$
```

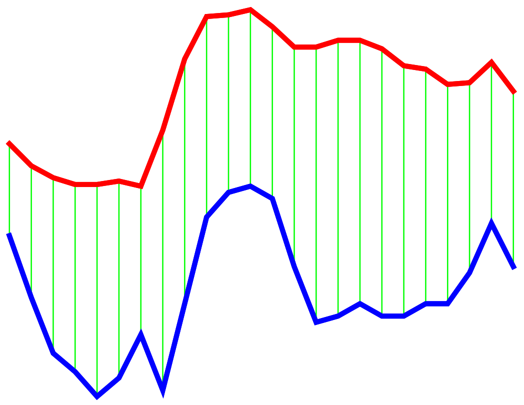
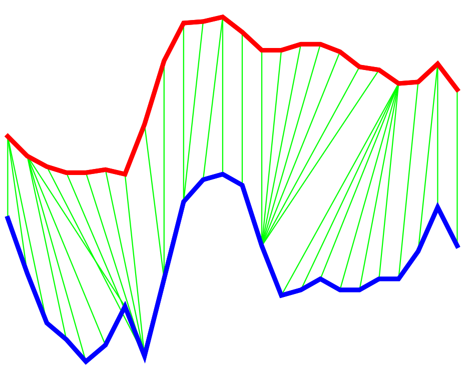


Figure 2.1: Illustration of Euclidean distance for time series

S and T. ED is the sum of all the point-wise differences (length of the green lines in Figure 2.1) between S and T. Assuming both S and T are normalised, Equation 2.1 describes the calculation of ED.

$$\mathrm{ED}(S,T) = \sum_{i=1}^{L} (S(i) - T(i))^2 \tag{2.1}$$

ED is a metric and satisfies the triangle inequality. This allows it to be used with indexing strategies [37, 38, 39, 40, 41, 42, 43] to support fast similarity searching. Interestingly, the Euclidean distance is extremely competitive with other distance measures when the dataset is large [22]. However due to its simplicity, ED is not robust to temporal misalignments and noise [22]. This then leads to the development of elastic distances such as Dynamic Time Warping (DTW) [45].


(a)

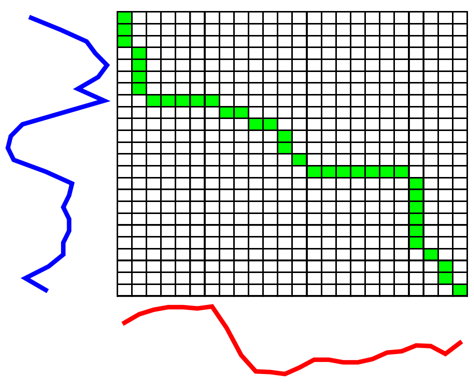

(b)

Figure 2.2: Example of (a) DTW alignment for two time series. (b) Cost matrix D_{DTW} with warping path $\mathcal{A}$ (green)

2.2.2 Dynamic Time Warping

Dynamic Time Warping (DTW) has been studied and tested extensively on the benchmark time series archive [6]. The NN-DTW algorithm was the leading classifier for TSC [17, 22, 23, 24]. DTW was firstly introduced as a spoken word recognition tool [45, 81] to handle distortions in the time axis, which could not be handled by the Euclidean distance. It stretches a time series and realigns the time series to better match the another time series [45]. Figure 2.2a illustrates the alignment of two time series S and T using DTW, as compared to the Euclidean Distance in Figure 2.1.

DTW has a complexity of $O(L^2)$ and finds the optimal alignment (*warping*) path along a $L \times L$ cost matrix $\mathcal{D}_{\text{DTW}}$ using dynamic programming. Each cell of the matrix $\mathcal{D}_{\text{DTW}}(i,j)$ represents the cumulative cost of aligning the two time series as described in Equation 2.2.

$$\mathcal{D}_{\text{DTW}}(i,j) = (S(i) - T(j))^2 + \min \begin{cases} \mathcal{D}_{\text{DTW}}(i-1,j-1) \\ \mathcal{D}_{\text{DTW}}(i,j-1) \\ \mathcal{D}_{\text{DTW}}(i-1,j) \end{cases} \tag{2.2}$$

An example of the optimal *warping path* is illustrated as the green path in Figure 2.2b. The *warping path* of S and T is a sequence $\mathcal{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_P \rangle$ of *links*. Each link is a pair

$\mathcal{A}_k = (i, j)$ indicating that S_i is aligned with T_j. $\mathcal{A}$ must obey the following constraints:

- **Boundary Conditions**: $\mathcal{A}_1 = (1, 1)$ and $\mathcal{A}_P = (L, L)$.
- **Continuity**: for $\mathcal{A}_k = (i_k, j_k)$ and $\mathcal{A}_{k+1} = (i_{k+1}, j_{k+1})$, $i_{k+1} - i_k \leq 1$ and $j_{k+1} - j_k \leq 1$
- **Monotonicity**: for $\mathcal{A}_k = (i_k, j_k)$ and $\mathcal{A}_{k+1} = (i_{k+1}, j_{k+1})$, $i_{k+1} - i_k \geq 0$ and $j_{k+1} - j_k \geq 0$

Finding this optimal warping path can be very time consuming especially for long time series. Hence, it is common to apply a global constraint to the path such that $S(i)$ and $T(j)$ can only be aligned if they are within a window range [81, 82, 83]. This limits the points in T that $S(i)$ can be aligned [27, 45]. This is known as Constrained DTW. Furthermore, a global constraint also reduces pathological warping that will often reduce the classification accuracy [83, 84]. There are many variants of this constraint such as the Ratanamahatana-Keogh Band [83], Itakura Parallelogram [82], and the most widely adopted Sakoe-Chiba Band [81].

This thesis will only focus on the Sakoe-Chiba Band which is commonly known as the warping window, w [45, 81] and this is written as DTW$_w(S, T)$. Note that $0 \leq w \leq L-1$ where DTW$_0$ corresponds to the Euclidean distance and DTW$_{L-1}$ is equivalent to unconstrained DTW. Figure 2.3 shows an example with warping window $w = 3$, where the alignment of two time series is constrained to be inside the gray band (see Figure 2.2b for unconstrained).

2.2.3 Derivative Dynamic Time Warping

The Derivative Dynamic Time Warping (DDTW) is a variant to DTW [84]. It aims to reduce singularities by transforming the time series into first order derivative [84]. Singularities arise for example when a point on a rising trend is mapped to a point on a falling trend, which is counter-intuitive [84]. First order derivative eliminates this problem by considering the higher level feature of shape rather than just the value of the data-point in the time series [84]. The derivative for a time series $S = \{S(1), , ..., S(L)\}$ is $S' = \{S'(2), ..., S'(L-1)\}$ and is computed using Equation 2.3. Here, $S'(i)$ is defined as the average of the slopes between $S(i-1)$, $S(i)$ and $S(i+1)$ [84]. Note that, $S'(i)$ is not defined for the first and last element of the time

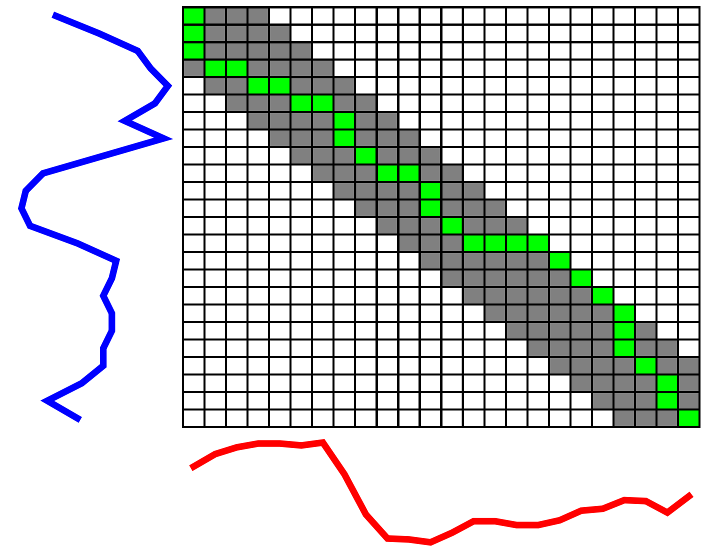


Figure 2.3: DTW with warping window, $w = 3$

series. Using the transformed time series, DDTW is computed the same way as DTW with dynamic programming [84]. Similarly a warping window w can be applied to DDTW to better reduce pathological warping and speeds up DDTW computation.

$$S'(i) = \frac{(S(i) - S(i-1)) + (S(i+1) - S(i-1))/2}{2} \tag{2.3}$$

2.2.4 Weighted Dynamic Time Warping

The Weighted Dynamic Time Warping (WDTW) is another variant of DTW that is proposed to reduce pathological warping and singularities [85]. Instead of using a warping window to prevent the alignment of $S(i)$ with $T(j)$ that are too far away, WDTW weights the cost of aligning $S(i)$ to $T(j)$ using a modified logistic weight function [85]. The intuition is that, if $S(i)$ is far away from $T(j)$ in the time dimension, i.e. i is far from j, it will have a larger weight and vice versa. Equation 2.4 describes this weight function where $\mathtt{W}_{max}$ is the upper bound for the weights and is typically set to 1 [85]. The parameter g controls the level of penalization for further points as illustrated in Figure 2.4 [85]. The optimal range for g should be distributed between 0.01 to 0.6 as suggested by the authors [85]. Furthermore, weights can also be applied to DDTW – giving the Weighted DDTW distance [85].

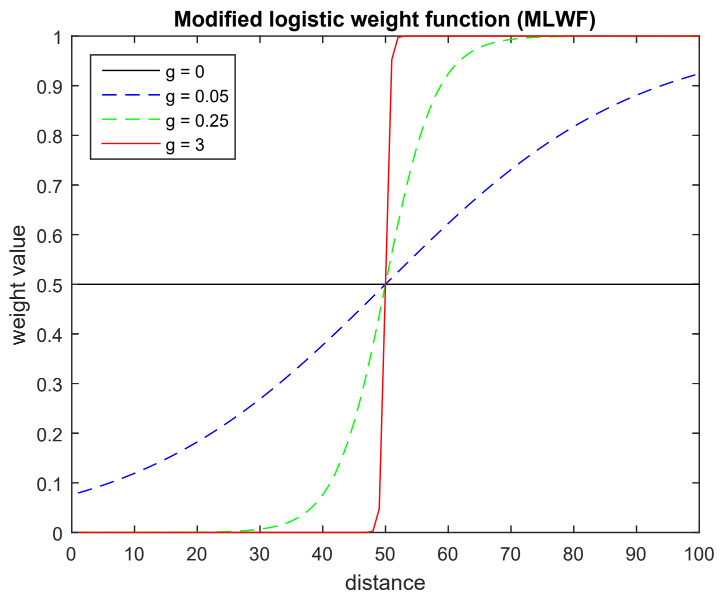


Figure 2.4: Modified logistic weight function for WDTW from [85]

$$\mathtt{w}_a = \frac{\mathtt{W}_{\max}}{1 + e^{-g \cdot (a - L/2)}} \tag{2.4}$$

2.2.5 Longest Common Subsequence

The Longest Common Subsequence (LCSS) is an edit distance like DTW. It is commonly used for pattern matching in the context of string sequences [86, 87]. It finds the longest common subsequence that best matches the two string sequences. For example, the green lines in Figure 2.5 shows the matching of these two sequences where the best subsequence is 'TABCFD' with a length of 6.

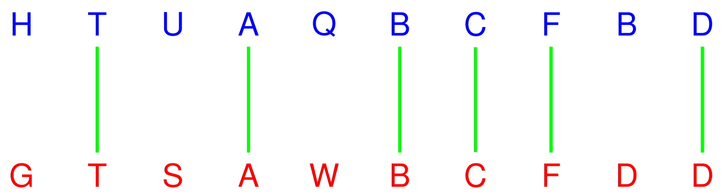


Figure 2.5: Example of string matching with Longest Common Subsequence. Green line indicates a match between two characters.

Although developed for string sequences, LCSS can be extended to numeric sequences (time series) using a distance threshold ε. Two elements are considered a match if the distance between them is less than ε. LCSS is also computed using a cost matrix $\mathcal{D}_{\text{LCSS}}$, where each cell of the matrix $\mathcal{D}_{\textsf{LCSS}}(i,j)$ indicates the number of matches between the two time series (length of the longest common subsequence), described in Equation 2.5. A global constraint can also be applied to LCSS and is commonly known as Δ [17].

$$\mathcal{D}_{\textsf{LCSS}}(i,j) = \begin{cases} 0 & \text{if } i=0, j=0 \\ 1 + \mathcal{D}_{\textsf{LCSS}}(i-1,j-1) & \text{if } |S(i)-T(j)| \le \varepsilon \\ \max \begin{cases} \mathcal{D}_{\textsf{LCSS}}(i-1,j) \\ \mathcal{D}_{\textsf{LCSS}}(i,j-1) \end{cases} & \text{otherwise} \end{cases} \tag{2.5}$$

2.2.6 Edit Distance with Real Penalty

Most of the edit distances like DTW and LCSS are not metric and do not satisfy the triangular inequality. Being a metric is important for indexing the training set, which can be used for fast nearest neighbour retrieval. DTW does not satisfy the triangular inequality because it replicates the previous element when a gap is added. The Edit Distance with Real Penalty (ERP) on the other hand, is an edit distance that satisfies the triangular inequality [88, 89]. ERP uses the distance between the two points as the penalty cost if a gap is not added. If a gap is added, the penalty will be the distance between that point and a constant penalty parameter g [88]. This is described in Equation 2.6, where ERP is computed with dynamic programming using a cost matrix $\mathcal{D}_{\textsf{ERP}}$. Similarly the alignment path can also be constrained with the `bandsize` parameter to prevent pathological alignment.

$$\mathcal{D}_{\textsf{ERP}}(i,j) = \min \begin{cases} \mathcal{D}_{\textsf{ERP}}(i-1,j-1) + (S(i)-T(j))^2 \\ \mathcal{D}_{\textsf{ERP}}(i-1,j) + (S(i)-g)^2 \\ \mathcal{D}_{\textsf{ERP}}(i,j-1) + (g-T(j))^2 \end{cases} \tag{2.6}$$

2.2.7 Move-Split-Merge Distance

Previous distance measures have a few limitations. The L_p-norm distances are not robust to temporal misalignment. Edit distances like DTW and LCSS are not metric, which prevent them from being used in various data mining tasks such as indexing, clustering and visualisation [90]. Although ERP is a metric, but it is not translation invariant. In other words, the similarity ranking can be altered radically by changing the origin of the coordinate system [90]. Moreover, ERP does not treat all the values equally, i.e. it prefers to delete and insert values close to zero [90]. Therefore, the Move-Split-Merge (MSM) distance is proposed to satisfy this set of properties, that no other existing distance measures can satisfy [90].

The MSM distance uses a set of operations (*Move, Split, Merge*) to transform a time series to better match the other one [90]. The cost for the *move* operation is the pairwise distance between two points [90]. The cost for the *split* and *merge* operations includes a constant penalty value c [90]. Equation 2.7 gives the cost function for MSM. Similar to DTW, MSM has a quadratic complexity of $O(L^2)$ and is computed with dynamic programming. Equation 2.8 describes the computation of the elements in the cost matrix $\mathcal{D}_{\text{MSM}}$ [90].

$$\mathcal{C}(S(i), S(i-1), T(j)) = \begin{cases} \text{c} & \begin{array}{ll} \text{if} & S(i-1) \leq S(i) \leq T(j) \\ \text{or} & S(i-1) \geq S(i) \geq T(j) \end{array} \\ \text{c} + \min \begin{cases} |S(i) - S(i-1)| \\ |S(i) - T(j)| \end{cases} & \text{otherwise} \end{cases} \tag{2.7}$$

$$\mathcal{D}_{\text{MSM}}(i,j) = \min \begin{cases} \mathcal{D}_{\text{MSM}}(i-1, j-1) + |S(i) - T(j)| \\ \mathcal{D}_{\text{MSM}}(i-1, j) + \mathcal{C}(S(i), S(i-1), T(j)) \\ \mathcal{D}_{\text{MSM}}(i, j-1) + \mathcal{C}(T(j), S(i), T(j-1)) \end{cases} \tag{2.8}$$

2.2.8 Time Warp Edit Distance

All the previous distance measures do not consider the timestamps of the time series. Timestamps are important when time series are sampled with various sampling rates that might be non-uniform [91]. The Time Warp Edit Distance (TWED) is proposed to compare time series using their timestamps [91]. Other than comparing time series with different sampling rate, considering the timestamps of the time series also allows the comparison of approximate (down-sampled) representation of the time series. It provides a useful relationship between the original time series and the down-sampled time series [91]. It is also computed with dynamic programming using a cost matrix $\mathcal{D}_{\mathsf{TWED}}$.

There are three operations (*delete*$_A$, *delete*$_B$ and *match*) in TWED that transform a time series for better matching [91]. A constant λ penalty is used in the *delete* operations. The match operation computes the distance of the current and previous data points. TWED controls warping in time series by multiplying the timestamps differences with a constant stiffness parameter v. $v = \infty$ means off-diagonal points of the cost matrix $\mathcal{D}_{\mathsf{TWED}}$ are not considered and is similar to the Euclidean distance, while $v = 0$ is similar to DTW [91]. The cost of all the three operations and the computation of each elements in the cost matrix $\mathcal{D}_{\mathsf{TWED}}$ are described in Equation 2.9 and 2.10 respectively. Note that $t_S(i)$ is the timestamps.

$$\begin{aligned} match: \quad \gamma_M = &\quad (S(i) - T(j))^2 + (S(i-1) - T(j-1))^2 + \\ &\quad v|t_S(i) - t_T(j)| + v|t_S(i-1) - t_T(j-1)| \\ delete_A: \quad \gamma_A = &\quad (S(i) - S(i-1))^2 + v|t_S(i) - t_S(i-1)| + \lambda \\ delete_B: \quad \gamma_B = &\quad (T(j) - T(j-1))^2 + v|t_T(j) - t_T(j-1)| + \lambda \end{aligned} \tag{2.9}$$

$$\mathcal{D}_{\mathsf{TWED}}(i,j) = \min \begin{cases} \mathcal{D}_{\mathsf{TWED}}(i-1, j-1) + \gamma_M & match \\ \mathcal{D}_{\mathsf{TWED}}(i-1, j) + \gamma_A & delete_A \\ \mathcal{D}_{\mathsf{TWED}}(i, j-1) + \gamma_B & delete_B \end{cases} \tag{2.10}$$

Algorithm 2: TRAINEE($\mathcal{T}$, C)

```
Data: $\mathcal{T}$: training data with size $N$
Data: C: set of NN classifier paired with an elastic distance measure
Result: $\mathcal{P}^\star$: best parameter value for each distances
Result: $bestAccuracy$: best LOO-CV accuracy for each distances
1  foreach $C_i \in C$ do
2      $bestNCorrect_i \leftarrow -1$
3      for $\mathcal{P}_p^i \leftarrow \mathcal{P}_1^i$ to $\mathcal{P}_M^i$ do
4          $nCorrect \leftarrow 0$
5          foreach $T \in \mathcal{T}$ do
6              NN $= C_i$.NNSEARCH($T, \mathcal{T} \setminus T, \mathcal{P}_p^i$)
7              if NN.class $=$ $T$.class then $nCorrect$++
8          end
9          if $nCorrect > bestNCorrect_i$ then
10             $bestNCorrect_i \leftarrow nCorrect$
11             $\mathcal{P}_i^\star \leftarrow \mathcal{P}_p^i$
12         end
13     end
14     $bestAccuracy_i \leftarrow bestNCorrect_i / |\mathcal{T}|$
15 end
```

2.2.9 Ensembles of Elastic Distances

These distance measures are paired with a NN classifier for classification. Each of them are not significantly different from each other in terms of classification accuracy [17] and the choice depends on the dataset of interest. However, combining them in an ensemble will reduce the variance, creating a more accurate classifier. The Ensembles of Elastic Distances (EE) is a meta-classifier that consists of 11 NN classifiers with various distance measures that is significantly more accurate than each of the individual NN classifiers [17]. The distance measures used in EE are (1) Euclidean distance; (2) Full DTW; (3) Constrained DTW; (4) Full Derivative DTW; (5) Constrained Derivative DTW; (6) Weighted DTW; (7) Weighted Derivative DTW; (8) Longest Common Subsequence; (9) Edit Distance with Real Penalty; (10) Move-Split-Merge Distance; (11) Time Warp Edit Distance.

As shown in their respective sections, each distance measure has one or two parameters that need to be learned [17] and learning of these parameters is done with leave-one-out cross validation (LOO-CV). The process involves finding the NN of each time series inside the training set across a set of M predefined values. Algorithm 2 describes the training process of EE using LOO-CV. The algorithm searches for the parameter that gives the highest LOO-CV accuracy for each of the NN classifiers. Line 2 initialises the best accuracy for a NN classifier in EE. Lines 3-15 show the typical procedure of doing LOO-CV for the NN classifier. Line 6 searches for the NN of a time series T from all the instances in the training set $\mathcal{T}$ except T itself with respect to the distance d_i. The label (class) of NN is the predicted label for T. The parameter that gives the highest accuracy is stored. Then for classification, EE weights the prediction from each of the NN classifiers using their LOO-CV accuracy [17]. EE is significantly more accurate than all its individual NN classifiers [17], making it the most accurate classifier in the time domain. It is also part of the most accurate HIVE-COTE [15].

2.3 Scalable Time Series Classification

The current research in time series classification [22, 23, 51] is lagging behind the demand of processing large datasets. Most research in TSC [10, 15, 16, 17, 21, 24, 26, 32, 83, 92] was tested with the standard benchmark time series datasets in the scale of thousands [6]. Classification performed using the state-of-the-art classifiers such as EE, Shapelet Ensemble (SE), COTE and HIVE-COTE are very computational demanding on these small datasets [6]. When it comes to large datasets, for example, the remote sensing dataset that contains a million time series instances [1], they become even more impractical. Apart from the high classification time, the state of the arts also have high training complexity. Training these ensemble-based classifiers is similar to training their individual classifier, where the learning algorithm has to repeatedly test the classifiers at different parameter values. Therefore, the task to scale up TSC boils down to scaling up the classification algorithms that constitute these ensembles.

Fast and scalable time series classification algorithms have been developed in the literature. Most of them are centred around scaling up NN classifiers [27, 32]. As mentioned in Section 2.2, the NN-DTW algorithm was the state of the art before the ensemble-based classifiers [15, 17, 21] were introduced. On the other hand, the bias of the NN algorithm vanishes in the limit when the dataset size increases, which makes it a very competitive classifier with large datasets. However, it is highly non-scalable because it has to compare the query sample to all the samples in the training dataset (see Algorithm 1), which is prohibitive for most large datasets. For instance, if given a dataset of size N and time series of length L, NN-DTW has a classification complexity of $O(N \cdot L^2)$ per query.

The simplest way of speeding up NN type classifiers is to use cheap $O(1)$ to $O(L)$ lower bound functions to prune off unpromising candidates and reduce the search space. The aim is to minimise the number of expensive $O(L^2)$ distance computations. Lower bounding has been successful in speeding up NN-DTW [27, 32]. Moreover, DTW lower bounds have been studied widely [27, 28, 29, 30, 31, 32]. In this work, lower bounds are used as the foundation to speed up NN classifiers. Section 2.4.1 will explain more on lower bounds.

Instead of a single lower bound, different lower bounds can be cascaded with increasing complexity to create a tighter lower bound while still having a complexity that is much less than $O(L^2)$ [32]. Besides, we can also early abandon the distance computation [32]. This is based on the observation that most of the computations in the DTW cost matrix, $\mathcal{D}_{\text{DTW}}$ are redundant, i.e. they have values larger than the distance between the query and the current nearest neighbour. Rakthanmanon *et al.* [32] introduce the UCR_SUITE algorithm to search trillions of time series subsequences under DTW using all these optimisation techniques. The PRUNEDDTW algorithm is proposed to prune off the cells in the cost matrix $\mathcal{D}_{\text{DTW}}$, that are guaranteed to not be part of the DTW warping path [93].

Approximate NN (ANN) algorithms are popular to speed up NN type classifiers as they reduces the search space [46]. Instead of the full training database, ANN only considers the instances that have the potential to be the nearest neighbour. ANN algorithms may not return the actual nearest neighbour, but could still return neighbours that have the same label as the nearest neighbour. Often, the approximate NN is good enough if the datasets is large, as it is more likely that the approximate NN has the same label with the actual NN. ANN algorithms systematically index the training dataset so that the query will be compared to just a few instances in the dataset. Indexing the training dataset prevents unnecessary comparison of unpromising candidates, or candidates that are too far from the query, thus saving the search time. Note that the indexing structure can also be used with an exact search [35, 36]. ANN algorithms can be divided into three categories, partitioning trees, hashing and graph techniques [46]. Partitioning trees method is more effective than the other two. This method works by partitioning the data into several subspaces using clusters or hyperplanes, which can then be used with a tree structure. The k-d tree [33, 94] is the most popular partitioning tree method, but it is only effective for low-dimensional data [46], thus it is not suitable for time series with hundreds and thousands of dimensions.

There are a few work done to improve k-d tree for high-dimensional data. Particularly, the multiple randomised k-d tree with a priority search has shown to be the most efficient [46, 95]. Although effective for high-dimensional datasets, it is still not suitable to time series as it partitions the data based on its dimensions. Note that the dimension of a time series data is its timestamps (data attribute). Partitioning time series data using timestamps will lose information on the ordering of data attribute. This prevents them from being used by distance measures that are paired with the NN classifier. Muja and Lowe [46] propose another partitioning trees method for high-dimensional data, the Priority Search K-means Tree (PSKMT) algorithm. PSKMT has a higher precision than the multiple randomised k-d tree [46]. It partitions the data using the K-means clustering algorithm and searches the tree with a priority queue. The authors show that PSKMT is about an order of magnitude faster than the previous ANN algorithms [46].

Locality Sensitive Hashing (LSH) [96, 97] is also known to be an effective ANN search algorithm for high-dimensional data. It is sometimes referred to as the c-Approximate r-Near Neighbour. The idea is that any two points in a high-dimensional space that are within a range r, will have the same hash value and hashed into the same bucket. Far points outside of the range cr will be hashed into a different bucket [97]. The performance of LSH highly depends on the hashing family and functions. Usually, multiple hash tables with various hash functions of the same family are used to ensure that the nearest neighbour and the query are at least in one of the buckets. LSH provides a theoretical guarantee on the search quality that can be done in near $O(1)$ time, but in practice, partitioning trees methods often outperform them, as shown in [46]. There are many LSH families, the most suitable one for time series is the Exact Euclidean LSH (E^{2}LSH) [96]. E^{2}LSH computes the hash value of a high-dimensional data using random projections. Consider a time series S with the length L, the hash function projects S from a L-dimension space onto a set of integer using Equation 2.11 [97].

$$h_{a,b}(S) = \frac{a \cdot S + b}{w} \tag{2.11}$$

where a is a L dimensional random vector, and b is chosen uniformly from the range $[0, w]$.

The random projection method has also been used to search for time series subsequence by first converting them into symbols [98, 99]. One major disadvantage of LSH algorithms is that, it requires high storage cost [46].

Lastly, nearest neighbour graph techniques are the least popular. They find the nearest neighbours using a graph structure where edges connect the objects in the dataset to its nearest neighbours [46]. Experiments in [100] show that k-NN graph can be competitive with randomised k-d trees. Despite the good performance of k-NN graph, constructing the graph itself is expensive, which leads to the construction of approximate k-NN graph [101].

Most of the indexing techniques mentioned in previous paragraphs are not suitable for time series matching as they suffer from the curse of dimensionality [34]. Specific techniques have been developed for time series data by having a higher level representation of the time series [22] and use some kind of lower bounds for similarity search [35, 36, 37, 38, 39, 40, 42, 43, 102]. This is commonly known as the GEMINI Framework [39].

Agrawal *et al.* [102] are the first to index time series datasets for time series matching. They index time series using Discrete Fourier Transform (DFT) and R^*-trees [102]. DFT is used because it preserves the Euclidean distance in time and frequency domains as proven in the Parseval's Theorem [102]. Similar technique is proposed in [39] to index time series subsequence. Singular Value Decomposition reduces the dimensionality of the time series for fast time series matching [43]. Piecewise Aggregate Approximation (PAA) segments the time series and represents each segment with the mean value [38, 40, 41]. Usually the number of segments is much less than the time series length which can then be used with multidimensional indexing structures [38]. Haar Wavelets together with R-tree are also used to index time series data [37]. The iSAX algorithm builds an indexing tree structure using symbolic representations of time series [36]. Then, raw time series are compared using ED or DTW at the leaf nodes [36]. iSAX2.0 is proposed to overcome the scalability issue in building the iSAX index [35].

The search space can also be reduced using the average time series in each of the classes of the training database [26]. The average time series is computed using DTW Barycenter Averaging (DBA) [47] and has shown to significantly reduces the classification time and improves the classification accuracy [26].

There are also few work done in scaling up other TSC algorithms as well as reducing the training time. Notably the *Proximity Forest* [67] (explained in Section 2.1.3) is proposed recently to speed up EE and has the potential to scale and outperform HIVE-COTE. Another recent work proposed c-RISE [103] that gives a contract training time for the Random Interval Spectral Ensemble (RISE) – a component of HIVE-COTE [15]. Giving a contract time to train a classifier allows the classifier to be trained within a short time and without compromising on the accuracy. Recall that the shapelet algorithm [56] is not scalable as there are many shapelet candidates to scan through. Rakthanmanon *et al.* [57] propose the Fast Shapelets algorithm for fast shapelets discovery. The Fast Shapelets algorithm represents the shapelets at a higher level representation using SAX words [57]. There are also some work done to scale up bag of word approaches. For instance, the SAX Vector-Space model (SAX-VSM) [62], an extension to the BOP algorithm, is competitive to NN-DTW with lower bounding and early abandoning [62]. Then the Bag-of-SFA-Symbols in Vector Space (BOSS-VS) algorithm was proposed to speed up the BOSS algorithm [104]. The authors [104] show that the BOSS-VS algorithm is more accurate and is multiple order of magnitude faster than NN-DTW.

It is important to realise that although all these works presented good results in scaling TSC and time series mining, most of them were designed for the Euclidean space. Besides, most of them were also tested with the standard benchmark time series datasets, where the largest dataset contains a total of 9,236 time series in both training and testing set [6]. The largest known publicly available dataset is the Phoneme dataset that contains 370 thousand time series [105, 106], but their method was not designed for scalable TSC [106]. They are not scalable enough for the one million SITS dataset. Although the iSAX2.0 algorithm has shown to index a billion time series, but it was done under the Euclidean distance, which is not suitable for the task of creating land-cover maps, where the phenomena of interests are usually modulated by weather (see Chapter 3 for more details). The authors [36] did suggest that it is possible to apply iSAX2.0 with DTW, but it was unclear as to how effective it will be. More importantly, these techniques do not tackle the problem of inefficient learning algorithm for NN classifiers.

2.4 Lower Bounds for Distance Measures

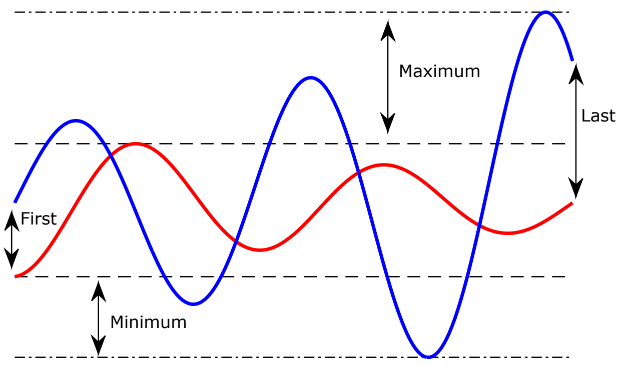
This section describes the existing lower bound functions for some of the distance measures. Lower bounds are very useful for speeding up NN type classifiers. In particular, lower bounds for DTW has been studied widely [27, 28, 29, 30, 31] and has been successful in speeding up NN-DTW algorithm [27, 32]. Lower bounds speed up NN classifiers by minimising the number of the expensive $O(L^2)$ distance computations and reducing the search space. Typically an effective lower bound has cheap $O(1)$ to $O(L)$ complexity. Algorithm 3 describes a NN search with lower bound. The main difference with a typical NN search (Algorithm 1) lies in line 3 where a lower bound is used before the actual distance computation to prune the candidates. Note that some lower bound functions require the distance function's parameter $\mathcal{P}$ to compute.

2.4.1 DTW Lower Bounds

The simplest and loosest DTW lower bound is the Kim lower bound (LB_KIM) [28] described in Equation 2.12. LB_KIM uses the maximum differences of the maximum, minimum, first and last points of S and T as the lower bound for DTW. With initialisation, LB_KIM can be computed very quickly with $O(1)$ time. Figure 2.6a illustrates this lower bound.

Algorithm 3: LBNNSEARCH($S, \mathcal{T}, \mathcal{P}$)

```
Input: $S$: Query
Input: $\mathcal{T}$: Data
Input: $\mathcal{P}$: Parameter for the distance
Result: $\mathrm{NN}$: Nearest neighbor of $S$ in $\mathcal{T}$
1 $\mathrm{NN}.\mathtt{dist} \leftarrow +\infty$
2 foreach $T \in \mathcal{T}$ do
3     if LOWERBOUND($S, T, \mathcal{P}$) $< \mathrm{NN}.\mathtt{dist}$ then
4         if DISTANCE($S, T, \mathcal{P}$) $< \mathrm{NN}.\mathtt{dist}$ then
5             $\mathrm{NN} \leftarrow T$
6         end
7     end
8 end
9 return $\mathrm{NN}$
```



(a)

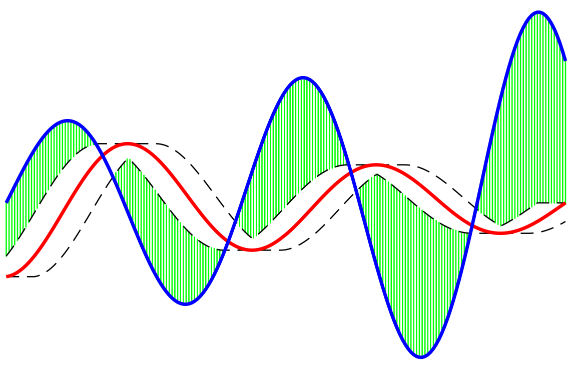

(b)

Figure 2.6: (a) KIM and (b) KEOGH lower bound

$$\text{LB\_KIM}(S,T) = \max \begin{cases} |S(1) - T(1)| \\ |S(L) - T(L)| \\ |\max(S) - \max(T)| \\ |\min(S) - \min(T)| \end{cases} \tag{2.12}$$

The Yi lower bound (LB_YI) [29] takes advantage that all the points in S that are larger than $\max(T)$ or smaller than $\min(T)$ must at least contribute to the final DTW distance. Thus the sum of their distances to $\max(T)$ or $\min(T)$ forms a lower bound for DTW as described in Equation 2.13.

$$\text{LB\_YI}(S,T) = \sum_{i=1}^{L} \begin{cases} (S(i) - \max(T))^2 & \text{if } S(i) > \max(T) \\ (S(i) - \min(T))^2 & \text{if } S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

The Keogh lower bound (LB_KEOGH) [27] is arguably one of the most used lower bound for DTW due to its simplicity and medium-high tightness. First, it creates two envelopes encapsulating the candidate time series. The upper envelope (UE) is built by finding the maximum value within a warping window w range, and the lower envelope (LE) is by finding the minimum, as shown in Equation 2.14.

$$\begin{aligned} UE_w^T(i) &= \max(T(i-w) : T(i+w)) \\ LE_w^T(i) &= \min(T(i-w) : T(i+w)) \end{aligned} \tag{2.14}$$

Then LB_KEOGH distance of S and T is the Euclidean distance of all the points in S that are outside of the envelope, to the envelope of T, UE_w^T and LE_w^T. Equation 2.15 describes the computation of LB_KEOGH. Figure 2.6b illustrates LB_KEOGH, where the sum of the length of the green lines is the LB_KEOGH distance. Note that LB_KEOGH$_w$ represents LB_KEOGH with warping window w.

$$\text{LB\_KEOGH}_w(S,T) = \sqrt{\sum_{i=1}^{L} \begin{cases} (S(i) - UE_w^T(i))^2 & \text{if } S(i) > UE_w^T(i) \\ (S(i) - LE_w^T(i))^2 & \text{if } S(i) < LE_w^T(i) \\ 0 & \text{otherwise} \end{cases}} \tag{2.15}$$

The Improved lower bound (LB_IMPROVED) [31] first computes LB_KEOGH and finds S', the projection of S on to the envelope of T using Equation 2.16, where $UE_w^T(i)$ and $LE_w^T(i)$ are the envelope of T. Then, it builds the envelope for S' and computes LB_KEOGH(T, S'). Finally, LB_IMPROVED is the sum of the two LB_KEOGH, described in Equation 2.17.

$$S'(i) = \begin{cases} UE_w^T(i) & \text{if } S(i) > UE_w^T(i) \\ LE_w^T(i) & \text{if } S(i) < LE_w^T(i) \\ S(i) & \text{otherwise} \end{cases} \tag{2.16}$$

$$\text{LB\_IMPROVED}_w(S,T) = \text{LB\_KEOGH}_w(S,T) + \text{LB\_KEOGH}_w(T,S') \tag{2.17}$$

LB_IMPROVED is tighter than LB_KEOGH, but has higher computation overheads, requiring multiple passes over the series. However, to be effective, it is usually used with an early abandon process, whereby the bound determined in the first pass is considered, and if it is sufficient to abandon the search, the expensive second pass is not performed.

The New lower bound (LB_NEW) [30] takes advantage of the boundary and continuity conditions for a DTW warping path to create a tighter lower bound than LB_KEOGH. The boundary condition requires that every warping path contains $(S(1), T(1))$ and $(S(L), T(L))$. The continuity condition ensures that every $S(i)$ is paired with at least one $T(j)$, where $j \in \{\max(1, i-w) \dots \min(L, i+w)\}$ and searched using binary search.

$$\text{LB\_NEW}_w(S,T) = (S(1)-T(1))^2 + (S(L)-T(L))^2 + \sum_{i=2}^{L-1} \min_{b \in T(i)} (S(i)-b)^2 \tag{2.18}$$

Instead of using standalone lower bounds, multiple lower bounds with increasing complexity can be cascaded, to form an overall tighter lower bound [32]. This greatly increases the pruning power and reduces the overall classification time. The UCR_SUITE [32] cascades LB_KIM, LB_KEOGH(S,T) and LB_KEOGH(T,S) to achieve a high speed up in time series search. Note that LB_KEOGH is not symmetric and by reversing the role of S and T can sometimes produce tighter bounds. It is important to realise that DDTW is a variant of DTW, so the lower bounds for DTW are directly applicable to DDTW.

2.4.2 ERP Lower Bounds

ERP is very similar to DTW, thus by taking into account ERP's penalty parameter g, DTW lower bounds are easily adapted for the ERP distance [88]. For instance, LB_KIM for ERP needs to consider that the first and last point may be a gap as described in Equation 2.19, where $S'(1) = S(1)$ or g, $S'(L) = S(L)$ or g. Then, $\max(\max(S), g)$ and $\min(\min(S), g)$ are used instead. The same applies to time series T.

$$\text{LB\_KIM}_{\text{ERP}}(S,T) = \max \begin{cases} |S'(1) - T'(1)| \\ |S'(L) - T'(L)| \\ |\max(\max(S), g) - \max(\max(T), g)| \\ |\min(\min(S), g) - \min(\min(T), g)| \end{cases} \tag{2.19}$$

To adapt LB_YI for ERP, the new minimum and maximum values are computed the same way as LB_KIM$_{\text{ERP}}$.

$$\text{LB\_YI}_{\text{ERP}}(S,T) = \sum_{i=1}^{L} \begin{cases} (S(i) - \max(\max(T), g))^2 & \text{if } S(i) > \max(\max(T), g) \\ (S(i) - \min(\min(T), g))^2 & \text{if } S(i) < \min(\min(T), g) \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Then to adapt LB_KEOGH for ERP, the envelopes need to be adjusted for g where the maximum and minimum values have to consider g, described in Equation 2.21. Note that `bandsize` is used instead of w. Then LB_KEOGH$_{\text{ERP}}$ is computed exactly the same way as LB_KEOGH for DTW using Equation 2.15 by substituting with the ERP envelopes.

$$\begin{aligned} UE^{T}_{\texttt{bandsize}}(i) &= \max(g, \max(T(i - \texttt{bandsize}) : T(i + \texttt{bandsize}))) \\ LE^{T}_{\texttt{bandsize}}(i) &= \min(g, \min(T(i - \texttt{bandsize}) : T(i + \texttt{bandsize}))) \end{aligned} \tag{2.21}$$

All the previous lower bounds are developed specifically for DTW. Thus, a new lower bound specifically for ERP is proposed [88]. By setting $g = 0$, LB_ERP is defined in Equation 2.22 as the absolute difference of the sum of both time series. The authors [88] show that LB_ERP has better pruning power than LB_KEOGH$_{\text{ERP}}$. Currently LB_ERP is only defined for $g = 0$ and there are no further proofs for $g \neq 0$.

$$\text{LB\_ERP}(S,T) = \left| \sum S - \sum T \right| \tag{2.22}$$

2.4.3 LCSS Lower Bound

The LCSS distance is computed using the percentage of points that are not a match – having distance larger than ε. Then, with the LCSS global constraint parameter Δ, lower bound for LCSS can be derived using similar idea as LB_KEOGH, i.e. constructing an envelope around the time series. This lower bound function is proposed in [107]. The envelope for T is constructed using ε and Δ as described in Equation 2.23.

$$\begin{aligned} \mathbb{UE}^T_\Delta(i) &= \max(T(i-\Delta) : T(i+\Delta)) + \varepsilon \\ \mathbb{LE}^T_\Delta(i) &= \min(T(i-\Delta) : T(i+\Delta)) + \varepsilon \end{aligned} \tag{2.23}$$

The sum of all $S(i) \in S$ within the envelope creates an upper bound (UB) to the longest common subsequence. Then the lower bound distance for LCSS (LB_LCSS) is $1 -$ UB, defined in Equation 2.24, as the percentage of points that are not within the envelope.

$$\text{LB\_LCSS}(S,T) = 1 - \frac{1}{L}\sum_{i=1}^{L} \begin{cases} 1 & \text{if } \mathbb{LE}^T_\Delta(i) \leq S(i) \leq \mathbb{UE}^T_\Delta(i) \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

Chapter 3

Time Series Indexing under Time Warping

In this chapter, we study the classification of large time series datasets using the Nearest Neighbour algorithm coupled with Dynamic Time Warping (NN-DTW). The NN-DTW algorithm is the core of time series classification (TSC) and has shown to be very competitive across various TSC algorithms [23, 24]. The NN algorithm compares the unknown time series to every other time series in a training database. With training database of N time series of lengths L, each classification requires $O(N \cdot L^2)$ computations. The databases used in almost all prior research have been relatively small (with less than 10,000 samples) and much of the research has focused on making DTW's complexity linear with L, leading to a runtime complexity of $O(N \cdot L)$. As we demonstrate with an example in remote sensing, real-world time series databases are now reaching the million-to-billion scale. This wealth of training data brings the promise of higher accuracy, but raises a significant challenge because N is becoming the limiting factor. As DTW is not a metric, indexing objects induced by its space is extremely challenging. We tackle this task in this chapter by developing TSI, a novel algorithm for Time Series Indexing which combines a hierarchy of K-means clustering with DTW-based lower-bounding. We show that on large databases, TSI makes it possible to classify time series 2-3 orders of magnitude faster than the state of the art.

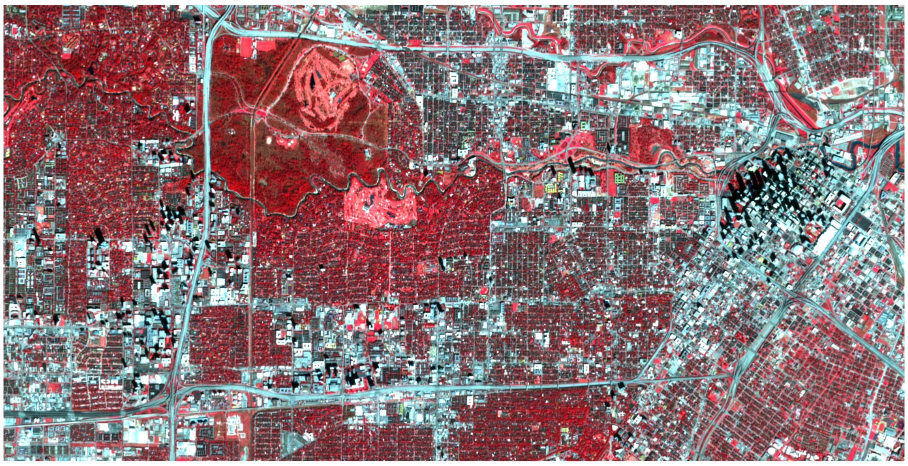


Figure 3.1: High-resolution image of Houston city near Westin Galleria taken by Sentinel-2A. © Copernicus Sentinel data [2016] for Sentinel data.

3.1 Introduction

The European Space Agency's Sentinel-2 satellites provide a full picture of Earth, every 5 days, at 10m resolution [7]. This and the corresponding NASA Landsat-8 programs introduce unprecedented opportunities to monitor the dynamics of any region of our planet over time and understand the constant flux that underpins the bigger picture of our world (more details at www.esa-sen2agri.org/SitePages/EOData.aspx). A high resolution satellite view of Houston city taken by the Sentinel-2A satellite, obtained from the Sentinels Scientific Data Hub (https://scihub.copernicus.eu/s2) is shown in Figure 3.1.

All images from these satellites are, by default, geometrically and radio-metrically corrected. Geometric corrections ensure that every pixel (x, y) always maps to the same geometric area. Radiometric corrections ensure that the spectral information is comparable from one image to the next of the series. This provides for each geographic coordinate on Earth, a time series of the "colours" that it underwent over the study period. One of the core tasks is to create temporal land-cover maps that describe the evolution of an area over time. This task is summarized in Figure 3.2: mapping the spectral evolution of a "pixel" (geographic coordinate) to a land-cover class such as "wheat crop", "broad-leaved tree" or "urban". Evolution is critical because, from space, all crops look the same; what makes it possible to correctly differentiate one from another is the temporal evolution (when the crop grows, when it is harvested etc.).

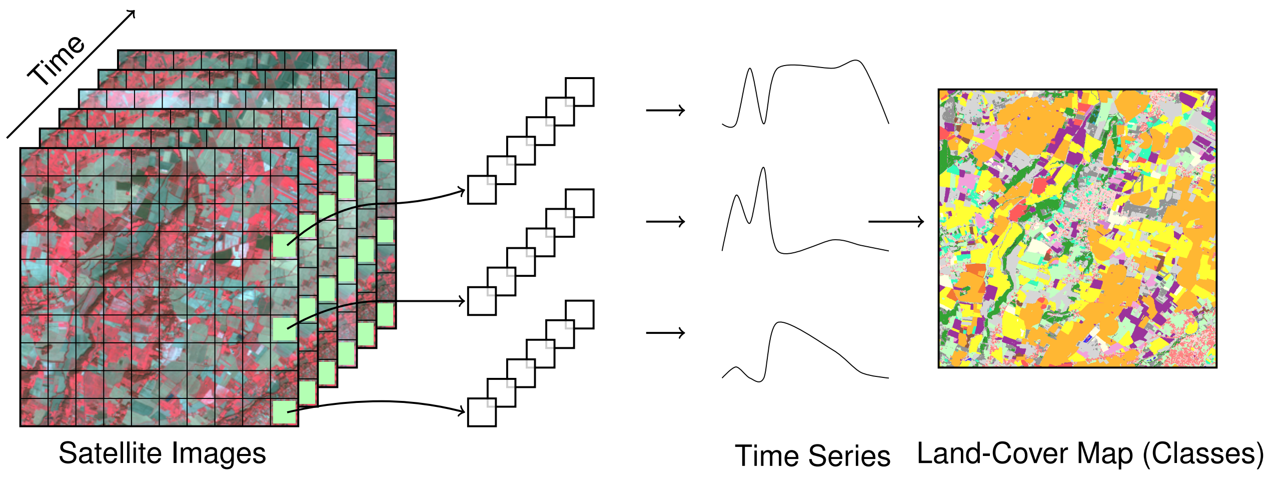


Figure 3.2: Production of a time series dataset from satellite image series.

Quite simply, research into time series classification lags behind the demands of modern space imagery, which produce terabytes of data each day. Why? Most research into time series classification has addressed datasets that hold no more than 10 thousand time series [6]. In contrast, the Sentinel-2 satellite provides over 10 trillion time series, capturing Earth's land surfaces and coastal waters at resolutions of 10 to 60m [7]. Although much research has gone into classifying remote sensing images, few studies have analysed time series extracted from sequences of satellite images.

The go-to time series classification method in terms of accuracy for this type of task is Nearest Neighbour coupled with Dynamic Time Warping (NN-DTW) [81, 108]. This is for two main reasons: (1) many phenomena of interest – vegetation cycles, for instance – have a periodic behaviour which can be slightly modulated by weather artifacts. These modulations result in distortions of canonical temporal profiles that are well handled by DTW [10]. (2) Time series are too short for Bag-of-word-type approaches [62, 104] to perform best.

NN-DTW cannot scale to the typical size of satellite datasets where it is common to have 100 million example time series [8, 9]. Even making the most of lower-bounding [27, 31], this is completely infeasible. Figure 3.3 illustrates this point: while all benchmark time series datasets [6] can be classified in less than 30 minutes, creating a temporal land-cover map for a city like Houston (16 million time series) using a bare minimum of 1 million training examples would take about a year. To create a land-cover map of Texas (7 billion time series) with 100 million samples would require 30k years of computation.

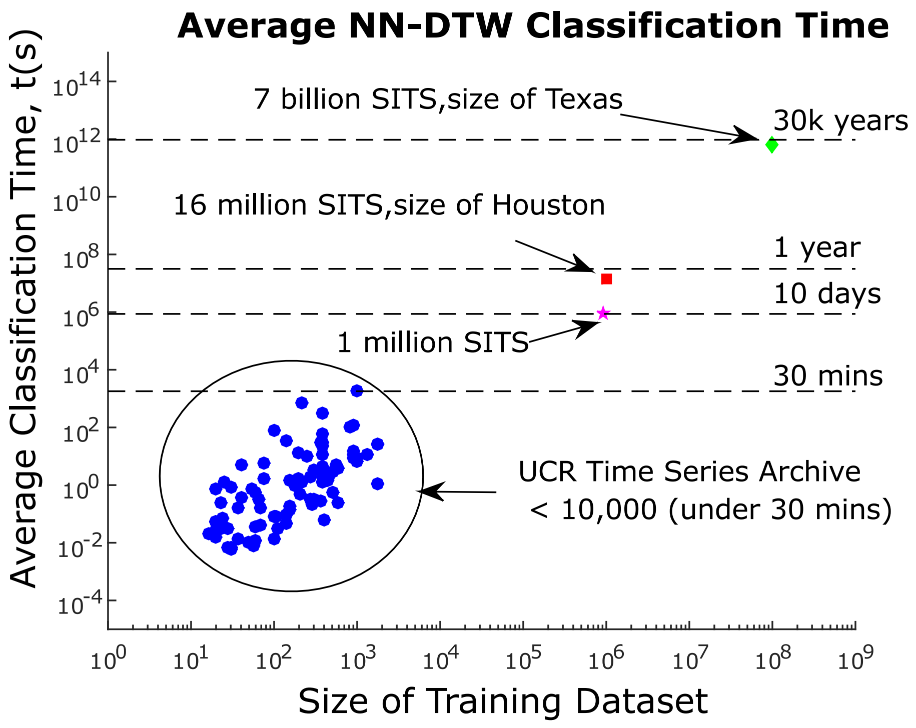


Figure 3.3: Average NN-DTW classification time on different datasets

With these motivations, this work tackles **Contract Time Series Classification**, where we would like to produce the most accurate classifier under a contracted time (obviously significantly smaller than running the NN-DTW). We propose a new algorithm that efficiently indexes the training database using a hierarchical K-means tree structure specifically designed for DTW. We will show that our algorithm reduces the time per query while retaining similar error to the state of the art, NN-DTW.

This chapter is organized as follows. In Section 3.2, we review some background and define the problem statement for our work. Then in Section 3.3 we introduce and describe our approach. Section 3.4 shows the empirical evaluation for our approach. Lastly, Section 3.5 offers some direction for future work and we conclude our work in Section 3.6.

3.2 Background and Motivation

3.2.1 Time Series Classification

Many time series classification algorithms in the literature such as Time Series Shapelets [56, 57], 1-NN BOSS [104] and SAX-VSM [62] have been shown to be competitive (and sometimes superior) to the state of the art, NN-DTW.

Nonetheless, as explained in the introduction,classification of the Satellite Image Time Series (SITS) is better tackled by NN-DTW. NN-DTW has been shown to be extremely competitive for many other applications [10, 16, 24, 26, 32, 83, 92]. It has been argued that the widespread utility of NN-DTW is due to time series data having autocorrelated values, resulting in high apparent but low intrinsic dimensionality. Experimental comparison of DTW to most other highly cited distance measures on many datasets concluded that DTW almost always outperforms other measures [24].

3.2.2 Nearest Neighbour Search

The nearest neighbour (NN) classifier (out of the context of time series) is one of the simplest classification algorithms but is nonetheless highly effective. It is a non-parametric, lazy learning algorithm and does not require any abstraction/training phase on the training dataset [109]. Its behaviour is also very interesting for large datasets, as its bias vanishes in the limit when the dataset size increases. k-NN classifier finds k nearest neighbours of the query in the training dataset and returns the dominating class as the class for the query sample [109]. Despite its nice behaviour in the limit, k-NN has the major drawback of not being scalable: it has to compare the query sample to all samples in the training dataset, which is infeasible for most large datasets.

An efficient and popular method of scaling up NN classification on large datasets is to use the k-d tree tree structure, where k is the dimensionality of the Euclidean search space [33]. k-d tree allows faster retrieval of the nearest neighbour to the query sample in a short amount of time. It is very efficient in the Euclidean space. However, it is not applicable for DTW-induced space as k-d tree tree partitions each dimension of the data recursively while DTW makes associations across dimensions. Because of that, k-d tree will not be applicable to NN-DTW for time series classification.

A large amount of research has been done to scale up NN-DTW for time series classification such as early abandoning [110] and lower bounding [27, 31, 32]. As DTW has time complexity of $O(L^2)$, these methods speed up classification by minimizing DTW computations. In this work, we focus on approximate nearest neighbour search, where it is typical to be two or more orders of magnitude faster than linear search [46].

Approximate nearest neighbour search algorithms index the data in the training dataset in a systematic manner so that the query sample will be compared to only a few promising candidates from the training dataset within a given time. It may not return the actual nearest neighbour, but with sufficient data it is highly likely that the approximate nearest neighbour will be of the same class.

The Priority Search K-means Tree (PSKMT) algorithm [46] is a recent breakthrough approach to nearest neighbour search that performs a priority search in a hierarchy of K-means clusterings. It has high precision on large high-dimensional datasets and is still one order of magnitude faster than previous approximate search algorithms [46, 111]. The algorithm outperforms existing approximate nearest neighbour algorithms on the 31 million sample SIFT dataset [46]. This algorithm offers an incredible opportunity for time series because, unlike k-d tree, it does not require that the data be tabular.

This work adapts and extends PSKMT to time series creating a novel efficient time series classification algorithm. Such adaptation was not possible before because K-means clustering was ill-defined for the DTW-induced space. Hence, we leverage our recent work on clustering time series consistently for DTW [26, 47] using DTW Barycenter Averaging (DBA) and adapt PSKMT to make the most out of the available lower bound for DTW. DBA has been extensively studied in [26, 47, 112] and a proof of convergence can be found in [26, 112].

Our experiments demonstrate that our method makes it possible to classify large time series datasets two orders of magnitude faster than the state of the art, NN-DTW with LB_Keogh [27, 113].

3.2.3 Contract Time Series Classification

In recent years there has been an increasing interest in using any-time algorithms for data mining [114, 115]. However, the variant known as contract algorithms have received less attention. Contract algorithms are a special type of any-time algorithms that require the amount of run-time to be determined prior to their activation. In other words, contract algorithms offer a trade-off between computation time and quality of results, but they are not interruptible.

Problem Statement Contract Time Series Classification: produce the most accurate time series classifier given (1) set constraints on computational resources available at classification time and (2) no constraints on computational resources at training time.

We assume that the computational resource constraint will be time, not space, and that it will be given to us in the form of the number of CPU cycles available for each query to classify. We assume that the constraint will be given as a positive integer $\mathcal{L}$ which is the number of time series to examine; for ease of exposition, we will report the result of our algorithm on different datasets at regular time intervals (i.e. with different $\mathcal{L}$).

3.3 Our Approach: DTW-Indexing of Time Series for Classification

Our algorithm, Time Series Indexing (TSI) is an adaptation of Priority Search K-means Tree (PSKMT) [46] to index time series embedded in a space induced by DTW. The general outline is as follows.

At **training time**, we construct a hierarchy of K-means clusterings over the training dataset; this prepares the indexing data structure that will allow fast querying at testing time. The K-means clustering is performed using DTW as the similarity measure for associating time series to their closest centroids. DBA [26, 47] is used to create and refine the centroids from the associated time series in the expectation phase.

At **testing time**, we maintain three priority queues. The first two queues store the potential branches to explore once exploration of the current branch is complete. The first stores those for which full DTW to the query has been calculated and the second stores those for which only lower bound have been computed. The third queue stores the nearest neighbours.

Since we prune off DTW with lower bound (LB), having 2 priority queues ensures that we always traverse from the actual closest branch without having to compute the full DTW distance for all potential branches. We start by descending the tree from the root to the first leaf, at each internal node following the branch closest to the query but pushing the alternatives to the priority queues. Those alternatives that can be excluded just on the lower bound go to the second queue while those for which the full DTW distance is computed go to the first queue. We explore the leaf, and then proceed to the closest branch that was not explored on the path to that leaf (stored as the head of the first queue). We continue in this cycle, stopping when we have exhausted our “contracted time” (or explored the full tree).

Algorithm 4 presents the algorithm for building the tree. The root node contains all the training data. The algorithm recursively clusters the data associated to each node into K clusters. A branch is formed leading to each of the K child nodes, each child node associated with the data in one cluster. The branch is labelled with the DTW average of the time series in the node to which it leads. The data associated with each child node is then clustered into K sub-clusters; and so on recursively. All nodes are labelled with the majority class of the data associated with them. This allows the algorithm to give a plausible class prediction even when we have not yet reach a leaf node. Every recursion is initialized using the standard non-deterministic K-means++ algorithm [116]. The recursion stops when a node contains K or fewer time series.

Algorithm 4: BUILDTREE($\mathcal{D}, K, I_{\max}, w$)

```
Input: $\mathcal{D}$: Time series dataset
Input: $K$: Branching factor
Input: $I_{\max}$: Maximum $K$-means iterations
Input: $w$: Warping window
Result: $\mathcal{T}$: Hierarchical $K$-means tree
if $|\mathcal{D}| \leq K$ then
    $\mathcal{T}$.createLeaf($\mathcal{D}$)
else
    // do $K$-means clustering
    $P \leftarrow$ kmeanspp($\mathcal{D}, w$)                    // initialize centroids
    for $iterations \leftarrow 1$ to $I_{\max}$ do
        $C \leftarrow$ ASSIGNTOCENTROIDS($P, \mathcal{D}, w$)          // cluster assignment
        foreach $C_i \in C$ do
            $P_i =$ DBA($P_i, C_i, w$)                        // update centroids
        end
    end
    foreach $C_i \in C$ do                                 // recursively build tree
        BUILDTREE($C_i, K, I_{\max}, w$)
    end
end
```

Algorithm 5: ASSIGNTOCENTROIDS($P, \mathcal{D}, w$)

```
Input: $P$: Cluster centroids
Input: $\mathcal{D}$: Time series dataset
Input: $w$: Warping window
Result: cluster: Clusters of time series
$cluster = \emptyset$
foreach $S_i \in \mathcal{D}$ do
    $nearest =$ LBNNSEARCH($S_i, P, w$)                         // see Algorithm 3
    $cluster[nearest]$.add($S_i$)
end
return $cluster$
```

To further speed up the clustering process, LB_KEOGH is used with NN-DTW in the LBNNSEARCH sub-routine to assign each time series to the nearest cluster centroid, as described in Algorithm 5. Our algorithm is not limited to just LB_KEOGH. It can be used with any DTW lower bounding functions such as LB_IMPROVED [31], depending on the application. In this work, we chose to use LB_KEOGH because in general, it performs well for most time series datasets.

Algorithm 6 describes the tree search algorithm. The tree is searched by first traversing down the tree to the first leaf node, outlined in Algorithm 7. At each level of the tree, the

Algorithm 6: SEARCHTREE($\mathcal{T}, S, \mathcal{L}, w$)

```
Input: $\mathcal{T}$: Hierarchical $K$-means tree
Input: $S$: Query time series
Input: $\mathcal{L}$: Number of time series to examine
Input: $w$: Warping window
Result: $k$-NN: $k$ nearest neighbours
1  Initialize priority queues & $seen = 0$
2  $W = $ envelope($S, w$)
3  TRAVERSETREE($\mathcal{T}, S, W, PQs, \mathcal{L}, seen, w$)
4  while ($PQs$ not empty) & ($seen < \mathcal{L}$) do
       /* find nearest branch                                     */
5      while $lbPQ$.top $<$ $dtwPQ$.top do
6          $branch = lbPQ$.dequeue
7          $d = \text{DTW}_w(S, branch$.centroid$)$
8          $dtwPQ$.enqueue($branch$)
9      end
10     if $dtwPQ$ not empty then
11         $\mathcal{T} = dtwPQ$.dequeue
12         TRAVERSETREE($\mathcal{T}, S, W, PQs, \mathcal{L}, seen, w$)
13     end
14 end
15 return $nnPQ$.data
```

algorithm proceeds with the nearest branch to the query time series. To efficiently search for the nearest branch, we first sort all the branches in ascending lower-bound distance to the query using Algorithm 8, then use NN-DTW to find the closest branch. This further minimizes DTW computations. The unexplored branches where DTW have been computed will be enqueued into the DTW priority queue while the remaining ones will be enqueued into the LB priority queue. These priority queues are implemented as min-heaps [117], with standard enqueue and dequeue functions.

When the query reaches a leaf node, the algorithm searches for the nearest time series using the same method as searching for the nearest branch. The k nearest time series found in the search so far are kept in the nearest neighbour priority queue, implemented as a max-heap. A max-heap allows faster retrieval of the k^{th} nearest neighbour from the query.

After exploring the leaf node, the algorithm proceeds to the next branch by dequeuing the DTW priority queue. Both DTW and LB priority queues are compared to ensure that the closest branch to the query is in the DTW priority queue: if the head of the LB priority queue is smaller than the head of the DTW priority queue, we dequeue that branch, com-

Algorithm 7: TraverseTree($\mathcal{T}, S, W, PQs, \mathcal{L}, seen, w$)

```
Input: $\mathcal{T}$: Hierarchical $K$-means tree
Input: $S$: Query time series
Input: $W$: Envelope for query time series
Input: $PQs$: $dtwPQ$, $lbPQ$ and $nnPQ$
Input: $\mathcal{L}$: Number of time series to examine
Input: $seen$: Time series seen so far
Input: $w$: Warping window
1  if $\mathcal{T}$ is leaf then
2      $T = \mathcal{T}$.data
3      $lbDistance$ = SortWithLB($W, T$)
4      foreach $T_i \in T$ do
5          $worstDistance = nnPQ.firstDistance$
6          if $lbDistance_i < worstDistance$ then
7              $d$ = DTW$_w(S, T_i)$
8              if $d < worstDistance$ then $nnPQ$.enqueue($T_i, d$)
9          end
10         if $++seen == \mathcal{L}$ then terminate
11     end
12 else
13     $C = \mathcal{T}$.children
14     $dtwFlag$ = [false, ..., false]
15     $bestSoFar = \infty$
16     $distances$ = SortWithLB($W, C$)
17     foreach $C_i \in C$ do
18         if $distances_i < best\_so\_far$ then
19             $distances_i$ = DTW$_w(S, C_i)$
20             $dtwFlag_i$ = true
21             if $distances_i < bestSoFar$ then
22                 $bestSoFar = distances_i$
23                 $C_q = C_i$
24             end
25         end
26     end
27     foreach $C_i \in C$ except $C_q$ do
28         if $dtwFlag_i$ then $dtwPQ$.enqueue($C_i, d_i$)
29         else $lbPQ$.enqueue($C_i, d_i$)
30     end
31     TraverseTree($C_q, S, W, PQs, \mathcal{L}, seen, w$)
32 end
```

pute its DTW distance and enqueue it into the DTW priority queue. The algorithm stops searching the tree when it has seen at least $\mathcal{L}$ time series from the leaf nodes. Here, $\mathcal{L}$ can also represent the "contracted" classification time. Our source code has been uploaded to Github[1].

[1]https://github.com/ChangWeiTan/TSI

Algorithm 8: SORTWITHLB($W, \mathcal{T}$)

```
Input: $W$, Upper and lower envelope for query
Input: $\mathcal{T}$, Set of time series
Output: $lbDistance$, Sorted LB distances
Output: $\mathcal{T}$, Sorted set of time series
1 $lbDistance = \emptyset$
2 foreach $S_i \in \mathcal{T}$ do
3     $lbDistance_i =$ LB_KEOGH$(W, S_i)$
4 end
5 sort$(\mathcal{T}, lbDistance)$
```

3.4 Empirical Evaluation

In this section, we comparatively assess the performance of our algorithm for large-scale time series classification against the state-of-the-art method:

- **LB_KEOGH NN-DTW**: This is the state-of-the-art approach of doing NN-DTW [27, 113] and serves as the baseline algorithm in this work. Time series are taken one by one in a random order: if their lower-bound distance to the query is greater than the best-so-far neighbour, then the time series is pruned and the method proceeds to the next series, else its actual DTW distance with the query is computed and it becomes the best-so-far neighbour if it is closer than the current best-so-far. Because the pruning power depends on how close are the neighbours found early in the search, we report the average results over 10 different shuffling of the data.
- **Time Series Indexing (TSI)**: This is our proposed method described in Section 3.3. As we use the non-deterministic K-means++ [116] initialization, we report the average results over 10 different runs.

Since our task is contract classification, we provide – for all methods – their results at different time intervals. It is worth noting that the results for all methods tend to the full NN-DTW as the time constraint tends to infinity. Our experimental datasets and results have been uploaded to [118]. Our experiments are divided into two parts:

A. First, we begin with a real world case study to show the practical utility of our technique.

B. Then, we assess the performance of the different methods on a diverse range of datasets. We show that our approach is more accurate than the conventional approach under a contracted time.

A. Satellite Image Time Series (SITS) Classification

As motivated in the introduction, the new-generation Earth Observation (EO) satellites have begun imaging Earth frequently, completely and in high-resolution. This introduces unprecedented opportunities to monitor the dynamics of any regions on our planet over time and revealing the constant flux that underpins the bigger picture of our world.

To the best of our knowledge, there has been little prior research into the classification of time series extracted from satellite image series – also called Satellite Image Time Series (SITS). The ability to monitor and classify these time series will have strong impact in many domains especially in the agriculture industry, marine applications and for environment monitoring. In prior work [10], we showed that DTW is a good measure for such time series, because of the non-linear distortions of prototypical ground surfaces, i.e., the fact that two neighbouring surfaces might have slightly different growth rates, although belonging to the same class of crop/tree.

In this work, we used 46 geometrically and radio-metrically corrected images taken by FORMOSAT-2 satellite. These images are corrected such that every pixel corresponds to the same geographic area on Earth. Each of these images consists of 1 million pixels and each pixel represents a geographic area of 64m^2, resulting in an area of 64km^2 per image. Each geographic area (x, y) ($\sim$pixel) in the image forms a time series with a length of 46, creating a dataset with 1 million time series. The series have been manually collected by experts in geoscience by a mix of photo-interpretation, ground campaigns and urban databases. All time series thus have a label about their temporal class such as “wheat”, “maize”, “broad-leaved tree”, etc. The formation of Satellite Image Time Series is illustrated in Figure 3.2, labelled with their temporal classes, represented in different colours.

SITS 1 Million	
Length, L	46
Size of Dataset, N	1,000,000
Number of classes	24
Warping window size	4
1-NN-DTW Error-Rate (10 fold cv)	0.168

Table 3.1: Properties of 1 Million SITS dataset

To ensure reproducibility of our results and encourage researchers to work on large time series datasets, we have obtained permission from the CesBIO and French Space Agency to make our satellite dataset available online in [118]; note that this is a very high-cost dataset (images are worth more than USD100,000 and collecting the labels required months of work) which we hope will be a significant motivation to the field. As there are no pre-defined train/test sets for this dataset, we used 10-fold cross validation results. In this experiment, a warping window of 4 was used: this is aligned with the phenology of observed phenomena for which similar stages of growth cannot be distant by more than about a month [10]. The properties of this dataset that will be used in this case study are shown in Table 3.1.

The case study was conducted by varying the contract time from the least to the more permissive, i.e. with more and more time allowed in the contract to perform the classification of each query until we have gone through the whole training dataset. We record the error for each query as we go through the whole training dataset. Note that at smaller time intervals, TSI was not able to predict the error because the algorithm has not reached a leaf node. In this case, we predict the error using the majority class of the time series set in the nearest branch explored to date.

We present our results in Figure 3.4 where the x-axis is in log-scale. The first element to note is that our algorithm, TSI is significantly better than the state of the art; having its curve consistently under the state of the art. Second, we can see that if we had a ‘contract’ to classify each time series in no more than 1ms, then TSI would obtain error rate of 0.195 as opposed to 0.374. The same observation holds for 0.1ms (error of 0.283 vs 0.5), 10ms (error of 0.178 vs 0.287) and 100ms (error of 0.17 vs 0.220).

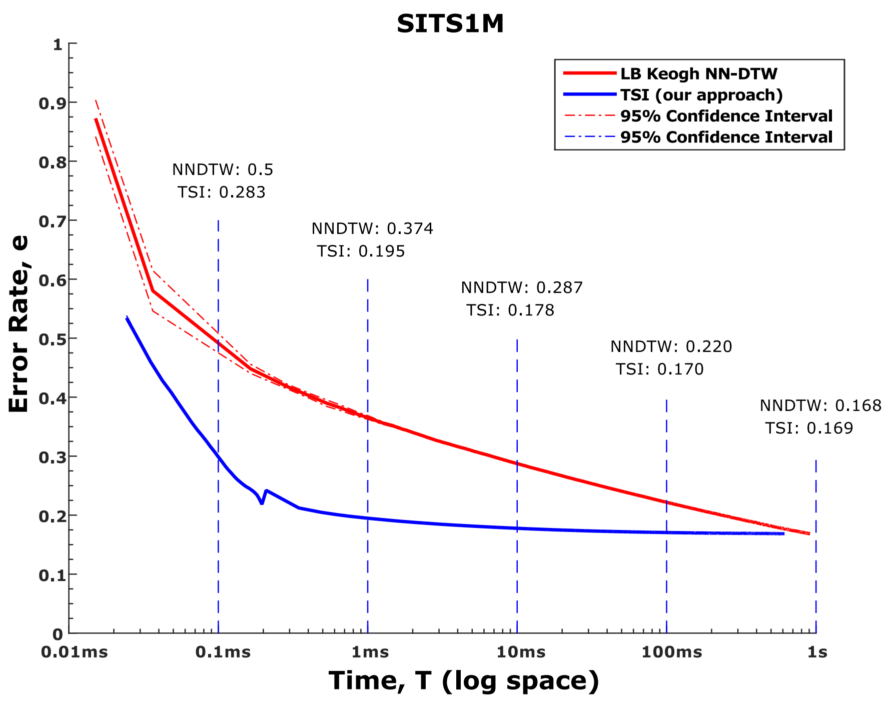


Figure 3.4: Comparative results on our 1 million SITS showing the average error rate per query as we go through the whole training dataset

This is a very important result: imagine if you were satisfied with an average error-rate of 0.2, then using our approach would classify each query within 1ms, as opposed to 500ms for the state-of-the-art approach. Having a million time series to classify, this would translate to our approach, TSI finishing in 17 minutes as opposed to 138 hours for LB_Keogh NN-DTW; a 500 times speed-up. It is only when getting to the far right of the curve that the overheads of TSI – linked to the exploration of the tree and maintenance of the priority queues – would become disadvantageous. This makes sense because if the contract is that you have the time to explore all of the training set, then you might as well just do that rather than using our approximate search.

Another point to note in Figure 3.4 is the slight jump in error for TSI at approximately 0.2ms. This is when the first leaf is expanded and the first class of an actual example is used in place of the majority class of a traversed branch.

B. Contract Time Series Classification

To show that our algorithm, TSI can classify more accurately within a contracted time, we run a statistical comparison of classifiers [119] on all the datasets from the standard UCR time series archive[2] [6] plus our SITS dataset, all together 85 datasets. We use the train/test split from [6] and warping window size reported in [6] for the time series archive. Note that, the datasets from the standard time series archive are relatively small, ranging from training size of 20 to 1,800 [6]. The different classifiers are compared using the Wilcoxon Signed-Ranks Test described in [119]. Wilcoxon Signed-Ranks Test is a test which ranks the differences in performance of two classifiers for each dataset [119]. We want to assess if 85 datasets is a large enough sample to show that our algorithm is statistically different.

Similar to the methodology in our case study, we record the error for each query at different time intervals, for every dataset and algorithm. To make the comparison fair for different datasets with different training size, we align the time intervals on the time of processing the whole dataset with LB_KEOGH NN-DTW. We consider 6 time intervals from 1% to 50% (of the time it takes LB_KEOGH NN-DTW to process one query). Similarly at smaller time intervals, we use the majority class of the nearest branch to predict the error. Let us take an example to ensure that our methodology is clear. We run LB_KEOGH NN-DTW for a dataset and found that it takes, on average, 1s to classify a query. We then study the error-rate of TSI at 10ms(1%), 100ms(10%), etc. up to 500ms(50%).

Using the error-rate at these time intervals, we calculate the difference in error-rate, d_i of LB_KEOGH NN-DTW and TSI on the i-th out of the N datasets. We then rank the differences by their absolute values at each time intervals [119]. Average ranks are assigned in case of ties [119]. These ranks are then used to calculate R^+ and R^- using Equation 3.1 and 3.2 respectively [119]. R^+ represents the sum of ranks for the datasets where the second algorithm outperforms the first and R^- the opposite [119]. In our context, the first algorithm refers to LB_KEOGH NN-DTW and the second refers to TSI.

[2]We exclude `ElectricDevices`, as most series are identical under time warping, thus preventing any clustering.

$$R^{+} = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{3.1}$$

$$R^{-} = \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{3.2}$$

With 85 datasets ($N = 85$), the critical value is calculated using Equation 3.3 that is distributed approximately normal [119]. To reject the null-hypothesis (where the two algorithms perform equally well) with $\alpha = 0.05$, the test statistic has to be less than the critical value, $z < -1.96$. The results are shown in the last column of Table 3.2 where we reject the null hypothesis highlighted in bold.

$$z = \frac{\min(R^{+}, R^{-}) - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{3.3}$$

Table 3.2 shows the results of the statistical comparison where we report the average rankings of LB_KEOGH NN-DTW and TSI across all datasets at the different time intervals and the Wilcoxon Test statistics. The average ranking allows us to identify the better performing classifier (emphasized in bold) if we reject the null hypothesis. In this case, our algorithm, TSI performs more accurately than the state of the art under a contracted time. The results show that TSI is more accurate than LB_KEOGH NN-DTW at all time intervals except for 1%, where we are unable to reject the null hypothesis. As the majority of datasets tested are small, the algorithms are unable to find even an approximate nearest neighbor at the 1% time interval, as a result of which they predict the majority class. Note that if the classes are very similar and the clusters do not well separate the classes, TSI can underperform LB_KEOGH NN-DTW. This is, for example, the case for the `Computers` dataset; the associated plot is available at [118].

LB_KEOGH NN-DTW vs TSI					
	Average Ranks		Wilcoxon Test Statistics		
Intervals	NN-DTW	TSI	R^+	R^-	z
1%	1.529	**1.471**	2034.5	1620.5	-0.907
10%	1.841	**1.159**	3449	206	**-7.105**
20%	1.871	**1.129**	3451	204	**-7.114**
30%	1.806	**1.194**	3219.5	435.5	**-6.099**
40%	1.741	**1.259**	2903	752	**-4.713**
50%	1.671	**1.329**	2616	1039	**-3.455**
Average	1.743	**1.257**			

Table 3.2: Wilcoxon test results

3.5 Optimizing the Number of Clusters, K

From the experiments, we observed that the number of clusters (branching factor), K is an important parameter that determines the convergence rate of the algorithm. In our experiments K was chosen to be 3. Although this is not the optimal K for the algorithm, we had shown that TSI still outperforms the state of the art if we just have 50% of the full 1-NN time.

However, we believe that for each dataset there exists an optimal K that allows the algorithm to converge faster to the full 1-NN error rate. This will be a trade-off between numerous factors. Larger K means that the length of each complete branch in the tree will be shorter and hence there will be fewer internal nodes to traverse to reach the leaf nodes that contain candidate nearest neighbors. However, it also means that at each internal node more DTW calculations must be performed to select the branch to follow.

There are many potential ways to optimize K. Here, we suggest an intuitive way to optimize K without compromising the error rate of the algorithm. We can vary K and record the average time per query to find the exact nearest neighbor at each K value. Then, we keep the K value that gives the minimum time per query.

3.6 Conclusion and Future Work

In this work, we have proposed the first algorithm to index DTW-induced space and showed that it is essential for the classification of time series when a large amount of data is available. We demonstrated that on a large remote sensing data where time series classification is critical, we are able to obtain the same accuracy up to 2 orders of magnitude faster than the state of the art, NN-DTW search; we can thus classify the entire 1 million dataset in 17 minutes instead of 5 days. This is extremely promising for larger remote sensing datasets that contain hundreds of millions of examples [8, 9].

Besides optimizing the branching factor, K, our future work will also include speeding up the search for the best warping window for large datasets and improved approaches for selecting a branch to follow. The current way of finding the best warping window, is to repeat LB_KEOGH NN-DTW with different warping windows, which is computationally expensive for large datasets. With the fast error convergence rate of our method, we can find the best warping window for large datasets in a short amount of time.

3.7 Acknowledgement

The authors would like to thank the researchers from CESBIO (D. Ducrot, C.Marais-Sicre, O. Hagolle and M. Huc) for providing the land cover maps and the geometrically and radiometrically corrected FORMOSAT-2 images. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-16-1-4023. This work was supported by the Australian Research Council under awards DE170100037 and DP140100087, and by the 2016 IBM Faculty Award (F. Petitjean). We would also like to thank Germain Forestier for his comments.

Chapter 4

Learning the Best Warping Window for Dynamic Time Warping Efficiently

In this chapter, we study the training time of the Nearest Neighbour algorithm coupled with Dynamic Time Warping (NN-DTW) on large datasets. The NN-DTW algorithm is the leading algorithm for time series classification and a component of the current best ensemble-based classifier for time series [15, 17, 21]. However, NN-DTW is only a winning combination when its meta-parameter – its warping window – is learned from the training data. The warping window (w) intuitively controls the amount of distortion allowed when comparing a pair of time series. With a training database of N time series of lengths L, a naïve approach to learning the w requires $O(N^2 \cdot L^3)$ operations. This often results in NN-DTW requiring days for training on datasets containing a few thousand time series only. In this paper, we introduce *Fast Warping Window Search* (FASTWWSEARCH): an *efficient* and *exact* method to learn w. We show on 86 datasets that our method is always faster than the state of the art, with at least one order of magnitude and up to 1,000x speed-up.

4.1 Introduction

Since its introduction in the 70s, Dynamic Time Warping (DTW) [81] has played a critical role for the analysis of time series, with hundreds (if not thousands) of papers published every year that make use of it. Many studies [16, 17, 21, 24, 32, 83, 92, 93] have shown that the One Nearest Neighbour Search with DTW (NN-DTW) outperforms most other algorithms when tested on the benchmark datasets [6], in spite of its simplicity.

In addition, the 2017 comprehensive benchmark of all time series classification methods [23] ranked COTE [21] as the most accurate classifier. COTE is an ensemble of classifiers – one of its base learners is NN-DTW with learned warping window (w). It directly follows that the complexity of NN-DTW is a lower bound on COTE's complexity. This becomes ever more problematic as the size of the training data increases. Figure 4.1 shows that the state-of-the-art method UCR_SUITE takes more than a day to learn the best w from 50,000 or more examples for this satellite image time series data. State-of-the-art methods LB_KEOGH [27] and PRUNEDDTW [93] pass the day threshold with just 20,000 time series. The blue curve shows that our Fast Warping Window Search (FASTWWSEARCH) method learns the best w in just 2 hours for 50,000 time series and can learn w for 100,000 time series in about 6 hours, a quantity of data that that is infeasible to process with the state of the art. With a training database of N time series of lengths L, a naive approach to learning the w requires $O(N^2 \cdot L^3)$ operations.

On the importance of the L^3 term. Figure 4.1 actually shows quite an optimistic picture, because this large dataset holds only very short series (with the length, $L = 46$), hence limiting the impact of the L^3 factor. For most datasets, L is typically ten times larger, which strongly influences runtime. We will see in Section 4.4 that speed-up can be up to 1,000x for datasets with longer series. Note that this dataset comes from the problem of creating a temporal land-use map from a series of satellite images [1, 10].

On the importance of the N^2 term. It has been shown that for a specific task, the larger the training data available, the smaller the warping window [44]. One could thus wonder if

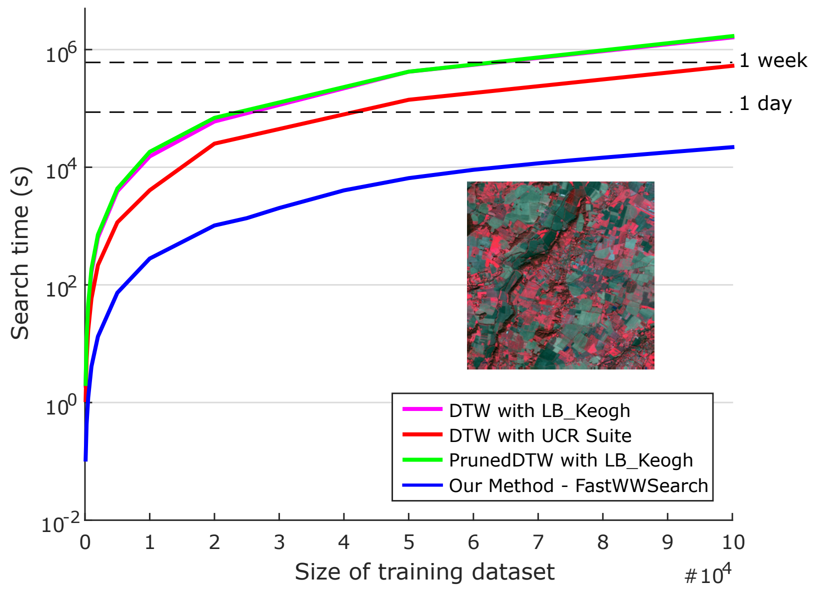


Figure 4.1: Training time for NN-DTW where the warping window is learned.

there is a need for a fast technique, because one could limit the scope of the search for the best warping window to a small subset of w values. However, for large datasets, the N^2 term takes over and becomes too important to even consider testing a few values of w.

In this work, we propose FASTWWSEARCH: a novel approach to speed up the learning process of the warping window for DTW. Our approach builds on the state of the art and introduces new bounds and exact pruning strategies with associated algorithms. FASTWWSEARCH is always at least one order of magnitude faster than state-of-the-art methods, and with speed-ups that can reach 1000x for some datasets. In essence, this algorithm takes a systematic approach to filling a vector representing the nearest neighbor for each w for each series in the training data. It searches efficiently and systematically to complete this vector, exploiting numerous bounds to avoid most computations. We release our code open-source to ensure reproducibility of our research, and to enable researchers and practitioners to directly use FASTWWSEARCH as a subroutine in further algorithms, including in the state-of-the-art methods for classification: COTE [21] and EE [17].

This chapter is organized as follows. In Section 4.2, we review some background and related work. Section 4.3 shows the intuition of our work and outlines our approach. Section 4.4 shows an evaluation of our method with the standard methods. Lastly, Section 4.5 concludes our work with some future work.

4.2 Background and Related Work

In this chapter, we use $\mathcal{T} = \{T_1, \cdots, T_N\}$ to denote a training dataset of size N where all time series are of length L, the letters S and T to denote two time series, and $T(i)$ to denote the i-th element of T.

4.2.1 Dynamic Time Warping

Dynamic Time Warping (DTW) was introduced in [45, 81]. As it has been presented numerous times in the literature and in Section 2.2.2, we simply define the elements that are the most critical for the understanding of this chapter and refer the reader to [27] for more information. DTW uses dynamic programming to find an optimal alignment of two time series S and T; it solves Equation 2.2 where a cell (i, j) of the cost matrix $\mathcal{D}_{\text{DTW}}$ accounts for all elements of S and T, up to i and j respectively. We then have DTW$(S, T) = \sqrt{\mathcal{D}_{\text{DTW}}(L, L)}$. Using dynamic programming, the alignment is solved in $O(L^2)$.

The warping path associated with DTW(S, T) is the sequence of minimum values taken consecutively by the cost matrix $\mathcal{D}_{\text{DTW}}(\cdot, \cdot)$. We note such warping path $\mathcal{A} = \langle \mathcal{A}_1, \cdots, \mathcal{A}_P \rangle$ and illustrate it on an example in Figure 2.2b. A couple $\mathcal{A}_k = (i, j)$ belonging to the warping path translates into an association $(S(i) - T(j))$ when aligned by DTW (illustrated in Figure 2.2a). Again here, as this has been covered in numerous papers, we refer the reader to [27] and Section 2.2.2 for more details about the warping path and its conditions (boundary, continuity and monotonicity).

4.2.2 Warping Window

A warping window w is a global constraint on the re-alignment that DTW finds (originally called Sakoe-Chiba band [81]), such that elements of S and T can be mapped only if they are less than w elements apart, and we write DTW$_w(S, T)$. Formally, this results in a warping path having as constraint $\forall (i, j) \in \mathcal{A}, |i - j| \leqslant w$. Figure 2.3 illustrates such a constraint with

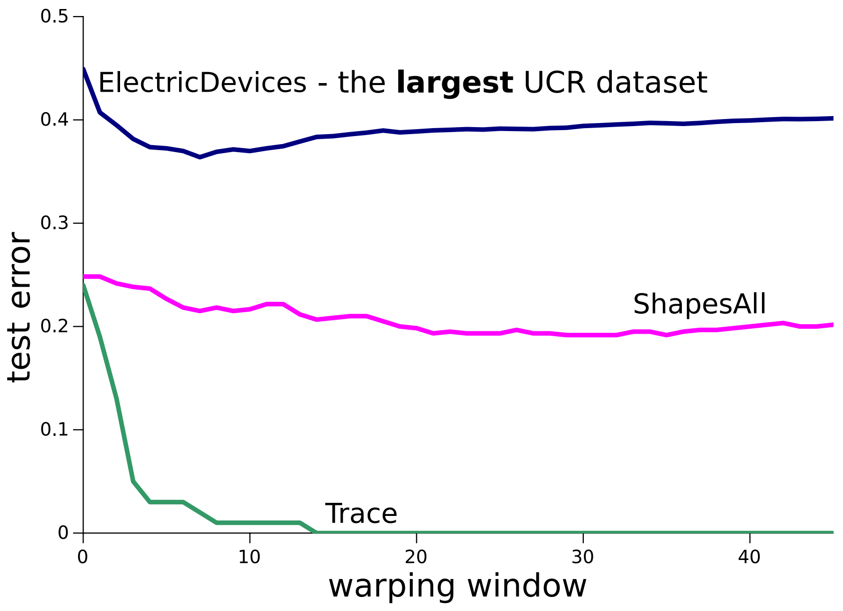


Figure 4.2: Test error on some datasets at various warping windows

$w = 3$ – the warping path is found within the gray band. Note that we have $0 \leqslant w \leqslant L - 1$, DTW$_0$ corresponds to the Euclidean distance, and DTW$_{L-1}$ is equivalent to unconstrained DTW. This added constraint has two main benefits: (1) preventing pathological alignments (and thus increasing the accuracy of the classifier) and (2) reducing the time complexity of DTW from $O(L^2)$ to $O(w \cdot L)$. Note that other types of constraints have been developed in the literature, including the Itakura Parallelogram [82] and the Ratanamahatana-Keogh band [83]. In this work, we focus on the warping window which, arguably, is by far the most used constraint in the literature [44, 92].

Why should the warping window be learned? The choice of the warping window (w) has long been known to have a strong influence on accuracy [6, 44, 92]. One of many examples is the `CinC_ECG_Torso` dataset [6], for which using a learned window reduces the error-rate from 35% to 7% [6]. We illustrate in Figure 4.2 the importance of learning the warping window on some datasets. In an extensive set of experiments Bagnall *et al.* [16] demonstrated that DTW is only competitive when the warping window is set via cross-validation. It is important to note that learning w can be critical even for large datasets. This is illustrated in Figure 4.2 with the largest dataset in the UCR archive – `ElectricDevices` – for which selecting an appropriate w reduces error by 7 percentage points relative to Euclidean distance (DTW$_0$).

Algorithm 9: SOTAWWSearch($\mathcal{T}$)

```
Input: $\mathcal{T}$: Data
Result: $w^\star$: The $w$ with lowest CV error
1  $bestErrors \leftarrow |\mathcal{T}| + 1$
2  for $w \leftarrow 0$ to $L - 1$ do
3      $errors \leftarrow 0$
4      foreach $S \in \mathcal{T}$ do
5          $nn \leftarrow$ LBNNSearch($S, \mathcal{T} \setminus \{S\}, w$)          // see Algorithm 3
6          if $nn$.class $\neq S$.class then
7              $errors \leftarrow errors + 1$
8          end
9      end
10     if $errors < bestErrors$ then
11         $w^\star \leftarrow w$
12         $bestErrors \leftarrow errors$
13     end
14 end
15 return $w^\star$
```

Finally, it is important to realise that the two state-of-the-art ensembles for time series classification – COTE [21] and EE [17] – include NN-DTW with learned warping window as one of their constituents. The time complexity of learning the w has become even more significant since Bagnall *et al.* [23] showed that COTE outperforms all existing methods for time series classification.

How do we learn the warping window? Learning of the warping window is not a simple problem; accepted methods in the field are all based on cross-validation: either on leave-one-out (LOO-CV) [16, 23] or on x-fold cross-validation [120]. In this work, we focus on LOO-CV for the sake of clarity, with all our algorithms directly usable for x-fold cross-validation. We illustrate the learning/search for LOO-CV in Algorithm 9.

4.2.3 Related Work

As learning the warping window involves leave-one-out cross-validation, the task boils down to being able to find the nearest neighbor of each time series in the training dataset (within the training dataset excluding themselves). We review below the state-of-the-art methods to perform this task efficiently.

Silva *et al.* [93] propose PRUNEDDTW to speed up DTW computation itself. They first compute an upper bound and skip the cells of the cost matrix $\mathcal{D}_{\text{DTW}}$ that are larger. The authors are able to learn the optimal window size faster than the naïve method [93]. However, as we will show in the experiments, the improvement for warping window search is only minimal.

Rakthanmanon *et al.* [32] propose the UCR_SUITE, which includes 4 optimization techniques: early abandoning, reordering early abandoning, reversing query and data roles in LB_KEOGH (see Section 4.2.4), and cascading lower bounds. Although they did not directly use their method to learn the warping window, it is natural to re-purpose it for this task.

However, as shown in Figure 4.1 (and described later in Section 4.4), those methods do not scale well for large datasets. This is mostly because they only try to tackle the impact of the length L on the $O(N^2 \cdot L^3)$ complexity. We will see that our FASTWWSEARCH method tackles both the impacts of L and N.

4.2.4 DTW Lower Bounds

Learning the warping window via cross-validation involves being able to efficiently find the neighbour of a time series within the training dataset, i.e. to perform a NN-DTW query for each time series. Lower-bounds to DTW have long been used in this context, they allow us to avoid the (expensive) DTW calculation if it is not needed. Several bounds have been introduced including LB_KIM [28], LB_KEOGH [27] (see Figure 2.6) and LB_IMPROVED [31]. Refer to Section 2.4 for more details. Lower bounds can also be used in cascade, starting by the looser (and computationally cheap) one and progressing towards tighter lower bounds [32]. Algorithm 3 shows a simple nearest neighbour search with lower bounds and can easily be modified for cascading lower bounds by changing the LB function in line 3.

4.3 Fast Warping Window Search for DTW

In this section, we introduce our approach: FASTWWSEARCH. In the first subsection, we start by introducing the mathematical properties that constitute the basis for our algorithm, which we introduce in the second subsection.

It is interesting to start by noting that, to find the best warping window via cross-validation, it is sufficient to know the nearest neighbour T of each time series S ($T \in \mathcal{T} \backslash S$) for each value of the warping window we want to test. The naïve (and almost state-of-the-art) algorithm for finding the best warping window is given in Algorithm 9. In this algorithm, we can observe that the loops are independent. The aim of this paper is to make the most of the inter-relation between the iterations of these two loops. As mentioned earlier, we focus our explanation on LOO-CV, but our method is directly extensible for any type of x-fold cross-validation.

4.3.1 Properties for FASTWWSEARCH

Property #1: Warping path can be valid for several windows

Theorem 1. *Let S, T be two time series, w_1 and w_2 two warping windows, and $\mathcal{A}_{w_1}$ and $\mathcal{A}_{w_2}$ their associated warping paths. $\mathcal{A}_{w_1} = \mathcal{A}_{w_2} \Rightarrow \text{DTW}_{w_1}(S,T) = \text{DTW}_{w_2}(S,T)$. In other words, $\text{DTW}(S,T)$ can only differ if the warping path differs.*

Proof. Let $\mathcal{A}_{w_1} = \langle (i_1^{w_1}, j_1^{w_1}), \cdots, (i_P^{w_1}, k_P^{w_1}) \rangle$, $\mathcal{A}_{w_2} = \langle (i_1^{w_2}, j_1^{w_2}), \cdots, (i_P^{w_2}, k_P^{w_2}) \rangle$. We have:

$$\begin{aligned} \text{DTW}_{w_1}(S,T) &= \textstyle\sum_{k=1}^{P} (S(i_k^{w_1}) - T(j_k^{w_1}))^2 && \text{Eq 2.2} \\ &= \textstyle\sum_{k=1}^{P} (S(i_k^{w_2}) - T(j_k^{w_2}))^2 && \text{(By hyp.)} \\ &= \text{DTW}_{w_2}(S,T) \end{aligned}$$

□

Theorem 2. *Let S, T be two time series, w a warping window, and $\mathcal{A}_w = \langle (i_1, j_1), \cdots, (i_P, k_P) \rangle$, then*

$$(|i_k - j_k| < w)_{\forall k} \Rightarrow \text{DTW}_w(S,T) = \text{DTW}_{w-1}(S,T)$$

In other words, if no point of a warping path 'touches' the extremity of the warping window at w, then the $\text{DTW}_w(S,T) = \text{DTW}_{w-1}(S,T)$*.*

Proof. By definition, $\text{DTW}_w(S,T)$ finds the warping path $\mathcal{A}_w$ such that the $\sum_{k=1}^{P}(S(i_k) - T(j_k))^2$ is minimized respecting constraint $(|i_k - j_k| \leqslant w)$. Our additional requirement $(|i_k - j_k| < w)$ ensures that $(|i_k - j_k| \leqslant w - 1)$. It results that the warping paths are equal, and by Theorem 1 so are the distances. □

In the following, we say that $\text{DTW}_w(S,T)$ has a "window validity" of $[\![v, w]\!]$ if all the warping paths $\mathcal{A}_w, \mathcal{A}_{w-1}, \cdots \mathcal{A}_v$ are the same and thus that the distances are all identical. We will see in Section 4.3.2 how we can use these theorem to prune many DTW computations, by starting with finding the NNs with larger warping window down to $w = 0$.

Property #2: DTW is monotone with w

Theorem 3. *Let S, T be two time series, and w a warping window, we have* $\text{DTW}_w(S,T) \leqslant \text{DTW}_{w-1}(S,T)$*.*

Proof. Reductio ad absurdum: Assume $\text{DTW}_w(S,T) > \text{DTW}_{w-1}(S,T)$, then this means that there exists a warping path $\mathcal{A}_{w-1}$ such that the associated cost is lower than the one for $\mathcal{A}_w$. This translates in DTW not having found the optimal solution at window w, which is a contradiction [81]. □

Figure 4.3 illustrates the combination of Theorem 1, 2 and 3, by showing the value of $\text{DTW}_w(S, T_{\{1,2,3\}})$ as a function of w. It shows that DTW decreases monotonically with w (Theorem 3), while the flat sections on the right section of the plot illustrate Theorems 1 and 2. Figure 4.3 also shows that the path computed for $w = 24$ remains valid down to $w = 1$ for T_1.

Property #3: LB_KEOGH is monotone with w

Theorem 4. *Let S, T be two time series, and w a warping window, we have* $\text{LB\_KEOGH}_w(S,T) \leqslant \text{LB\_KEOGH}_{w-1}(S,T)$*.*

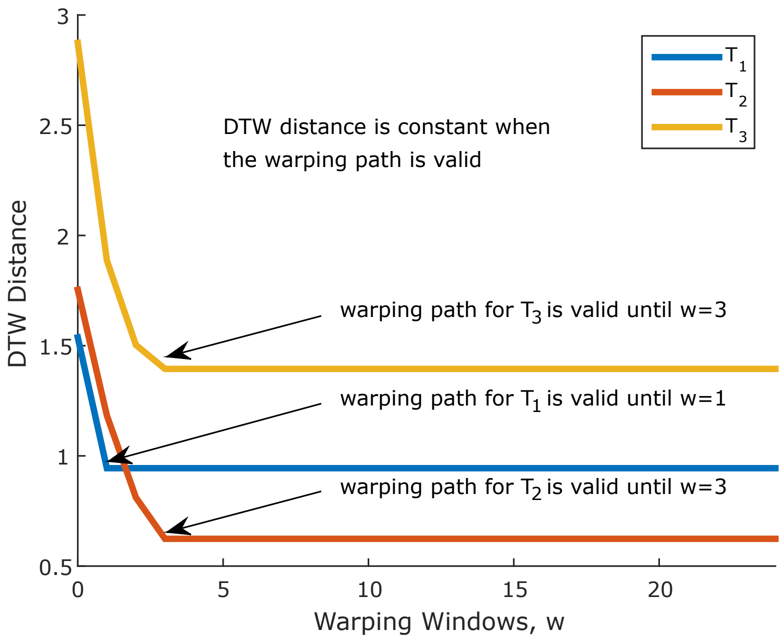


Figure 4.3: DTW distance at different w

Proof. Let us define UE^T and LE^T as the upper and lower envelopes for any time series T, such that the i^{th} element of the envelopes are defined as $UE^T_w(i) = \max(T(i-w) : T(i+w))$ and $LE^T_w(i) = \min(T(i-w) : T(i+w))$ [27]. Reductio ad absurdum using [27, Eq. 9]:

$$\text{LB\_K}{\small\text{EOGH}}_w(S,T) > \text{LB\_K}{\small\text{EOGH}}_{w-1}(S,T)$$
$$\Rightarrow \sum_{i=1}^{L} \begin{cases} U^T_w(i) & \text{if } S(i) > U^T_w(i) \\ L^T_w(i) & \text{if } S(i) < L^T_w(i) \end{cases} <$$
$$\sum_{i=1}^{L} \begin{cases} U^T_{w-1}(i) & \text{if } S(i) > U^T_{w-1}(i) \\ L^T_{w-1}(i) & \text{if } S(i) < L^T_{w-1}(i) \end{cases}$$

which contradicts the definition of lower and upper envelopes that gives $U^T_w(i) \leqslant U^T_{w-1}(i)$ and $L^T_w(i) \geqslant L^T_{w-1}(i), \forall i$. □

Intuitively, the smaller the window, the closer the envelopes are to the reference time series. This then results to a larger lower bound (the green part in Figure 2.6b). We will see in Section 4.3.2 that we use LB_KEOGH$_{w+1}$ as themselves lower bounds to LB_KEOGH$_w$.

	Nearest neighbor at warping windows				
	0	1	$\cdots$	$L-2$	$L-1$
T_1	$T_{24}(2.57)$	$T_{55}(0.98)$	$\cdots$	$T_{55}(0.98)$	$T_{55}(0.98)$
$\vdots$			$\vdots$		
T_N	$T_{60}(4.04)$	$T_{47}(1.61)$	$\cdots$	$T_{47}(1.61)$	$T_{47}(1.61)$

Table 4.1: Table of NNs for each w. A cell $(i, w) = T_k(dist)$ means T_i has T_k as its NN for window w with distance $dist$.

4.3.2 The FASTWWSEARCH Algorithms

We have presented the theoretical basis for our work; we now proceed with our algorithm.

Intuition behind FASTWWSEARCH Learning the warping window can be seen as creating a $(N \times L)$-table, illustrated in Table 4.1, giving the nearest neighbour (NN) of every time series for all windows. Once that table is filled, the best value of the window can be learned in one pass over it. We can frame the aim of FASTWWSEARCH as the construction of such an $(N \times L)$-table. Filling it naively, i.e. by computing DTW for each nearest neighbour NN(i, w) using DTW only, requires $O(N^2 \cdot L^3)$ operations. Note that, an exhaustive search of all L warping windows for large L and N can be extremely computationally demanding. Thus, most practitioners settle with a subset of w [17]. Our method applies to either all or a subset of the possible warping windows, simply starting from the largest and scanning through the windows down to the smallest.

Until recently, most of the research had focused on finding bounds for a fixed value of a warping window. Our method will explore bounds across columns of this table. Section 4.3.1 gives us two additional lower bounds that we use to prune potential nearest neighbors before we have to compute DTW$_w$:

Lower Bound name	Complexity
LB_KIM	$O(1)$
LB_KEOGH$_{w+k}$	$O(L)$
LB_KEOGH$_w$	$O(L)$
DTW$_{w+k}$	$O(w \cdot L)$

Note that we also use the fact that LB_KEOGH is not symmetrical in FASTWWSEARCH below. To take advantage of Theorems 3 and 4, we order our computations by *decreasing window size*. Our algorithm iterates from larger values of windows down to smaller ones. This has the consequence of obtaining either LB_KEOGH$_{w+k}$ or DTW$_{w+k}$ 'for free' when assessing window w (if pruning at step $w + k$ wasn't possible with LB_KIM), because those will have been calculated in a previous step with a higher w. We will see that these bounds should only be used if they have already been computed; there is indeed no point in computing DTW$_{w+k}$ to potentially prune DTW$_w$ of which the value is less expensive to compute.

Moreover, ordering our computation by decreasing window size also allows us to make the most of Theorem 1 and 2. Referring to Figure 4.3, the long flat tails correspond to large validity windows for DTW$_{L-1}$. It means that, in that flat section (and any subsequent other), no bounds are at all necessary, because we already know that the value of DTW (previously computed) has not changed. This element actually has two important consequences.

First imagine that we did find the nearest neighbor for a time series at window $w = L$; this implies that we had to calculate DTW$_L$ for those 2 series. If the warping path is valid down to $w = 0$, then a consequence of our Theorem 1, 2 and 3 is that we *know* that this will also be its nearest neighbor down to $w = 0$, and this without any additional calculations.

Second, when calculating an actual DTW$_w(S, T)$, even if the candidate T does not become the nearest neighbor of S at w, we know nonetheless that we do not need to recompute DTW$_{w'}(S, T)$ for all windows w' such that the warping path is valid (see Theorem 1 and 2).

Lazy Nearest Neighbor Assessment We now have all the elements necessary to the presentation of our FASTWWSEARCH algorithm. We start by presenting LAZYASSESSNN in Algorithm 10. LAZYASSESSNN is a function that, given a pair of time series (S, T), establishes if they can be less than a given distance d apart for a warping window w. It functions in a lazy fashion, by making the most of all possible bounds that we presented in Section 4.3.2. The algorithm tries lower bounds of increasing complexity until one of two things happen: (1) either a lower bound or DTW$_w(S, T)$ itself is greater than d, in which case the procedure aborts; or (2) we have DTW$_w(S, T) < d$, and we have actually calculated DTW$_w(S, T)$.

LAZYASSESSNN is lazy in that it postpones calculations for as long as it is possible to do so. As we will see next, one has to imagine here that LAZYASSESSNN will be called several times for the same pair of time series (S, T) for decreasing values of w. When w decreases, any value that was previously calculated for a larger window, becomes a lower bound for the current w. We use a cache $\mathcal{C}_{(S,T)}$ to store the previous results of LAZYASSESSNN obtained for larger w.

We first start by checking if the cache has been initialized; if not we compute LB_KIM, which is valid regardless of the warping window (line 1). On line 2, we then test where the cache last stopped, i.e. was it computing a lower bound for that target window w, was it computing a lower bound for another window $w' > w$, or was it computing an actual DTW distance (that might be still valid). We then assess if it last stopped having had calculated DTW for a larger window (lines 3–5); if that DTW$_{w'}$ has a path that is still valid and its value is smaller than d, then we return that value of the distance. We then assess if it had calculated DTW$_{w'}$ that is still valid and smaller than d. It is important to observe here that the code terminates when a distance is larger than d or DTW$_w$ is computed. If we cannot prune with DTW$_{w'}$, we proceed and check if we are able to prune using previously computed bounds (lines 6–7). Otherwise from lines 8 to 12, we use cascading lower bounds, testing if we can prune after each of them. Finally, if all the bounds fail to prune T, we compute DTW$_w$, store the results and if DTW$_w < d$, T is the new NN for S (line 13). We will see in our main algorithm that the next call to LAZYASSESSNN for the pair (S, T) will be with a smaller w.

Although our scheme *does* make it possible to use Early Abandon on LB_KEOGH [32] and to use LB_IMPROVED [31], we disregarded them after observing that they both increased computation time: early abandoning because it significantly increases the number of times the function has to be restarted; LB_IMPROVED because it requires to compute a projection of a series onto the other, which has an additional cost not justified by the added pruning power in our case.

We take the bounds ordered by computational complexity, which in practice usually correlates with tightness [32]. DTW$_{w'}$ will also generally be tighter than LB_KEOGH$_w$, especially when w' is close to w.

Algorithm 10: LAZYASSESSNN($\mathcal{C}_{(S,T)}, w, d, S, T$)

```
Input: $\mathcal{C}_{(S,T)}$: cache storing the previous measure between $S$ and $T$
Input: $w$: Warping window
Input: $d$: Distance to beat
Input: $S, T$: Time series to measure
Result: $\text{DTW}_w(S,T)$ if $\geqslant d$, else pruned
1  if $\mathcal{C}_{(S,T)} = \varnothing$ then $\mathcal{C}_{(S,T)} \leftarrow$ LB_KIM$(S,T)$
2  switch $\mathcal{C}_{(S,T)}$.stoppedAt do
       // LB calculated with larger window $w'$
3      case $\text{DTW}_{w'}$ do
4          if $w \in \mathcal{C}_{(S,T)}$.valid $\wedge$ $\mathcal{C}_{(S,T)}$.value $< d$ then
5              return $\mathcal{C}_{(S,T)}$.value
6          end
7      case LB_KIM or LB_KEOGH$_{w'}$ do
8          if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
9      end
       // Cascading LB_KEOGH and DTW
10     otherwise do
11         $\mathcal{C}_{(S,T)} \leftarrow$ LB_KEOGH$_w(S,T)$
12         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
13         $\mathcal{C}_{(S,T)} \leftarrow$ LB_KEOGH$_w(T,S)$
14         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
15         $\mathcal{C}_{(S,T)} \leftarrow \text{DTW}_w(S,T)$
16         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
17         return $\mathcal{C}_{(S,T)}$.value
18     end
19 end
```

Our core FASTWWSEARCH algorithm We now turn to our core algorithm: FASTWWSEARCH. Let us recall that the central aim of our algorithm is to build an $(N \times L)$-table that contains the nearest neighbour of each time series (out of N) and for each value of the warping window (out of L) – illustrated in Table 4.1. Once this table has been calculated, one pass over it is sufficient to determine the best value of the warping window. That pass is the entry point to FASTWWSEARCH and is presented in Algorithm 11. It assumes there exists a method to fill the table and returns the warping window with the lowest leave-one-out cross-validation error on $\mathcal{T}$.

The result of Algorithm 11 is identical to the state-of-the-art presented in Algorithm 9 – obviously assuming that the $(N \times L)$-table is calculated correctly, which is illustrated with our fail-safe experiments in Appendix A.

Algorithm 11: FASTWWS($\mathcal{T}$)

```
Data: $\mathcal{T}$: Training data
Result: $w^\star$: Best warping window
1  $\mathrm{NNs} \leftarrow$ FASTFILLNNTABLE($\mathcal{T}$)
2  $bestErrors \leftarrow |\mathcal{T}| + 1$
3  for $w \leftarrow 0$ to $L - 1$ do
4      $errors \leftarrow 0$
5      foreach $T_t \in \mathcal{T}$ do
6          if $\mathrm{NNs}[t][w]$.class $\neq T$.class then $errors{+}{+}$
7      end
8      if $errors < bestErrors$ then
9          $w^\star \leftarrow w$
10         $bestErrors \leftarrow errors$
11     end
12 end
```

Obviously, the core of our approach resides in how we calculate this table, which we present in Algorithm 12. At the highest level, our algorithm works by building this table for a subset $\mathcal{T}' \subseteq \mathcal{T}$ of increasing size, until $\mathcal{T}' = \mathcal{T}$. We start by building the table for $TSet'$ comprising only 2 first time series T_1 and T_2, and fill this $(2 \times L)$-table as if $TSet'$ was the entire dataset. At this stage it is trivial that T_2 is the nearest neighbour of T_1 and vice versa. We then add a third time series T_3 from $\mathcal{T} \setminus \mathcal{T}'$ to our growing set $\mathcal{T}'$. At this point, we have to do two things: (a) find the nearest neighbor of T_3 within $\mathcal{T}' \setminus \{T_3\} = \{T_1, T_2\}$ and (b) check if T_3 has become the nearest neighbor of T_1 and/or T_2. We can then add a fourth time series T_4 and so on until $\mathcal{T}' = \mathcal{T}$.

We now describe Algorithm 12 line by line. We start by initializing the NNs $(N \times L)$-table to $(\_, +\infty)$, which means that the table is empty and the distances are thus $+\infty$ (line 2). We then initialize $\mathcal{T}'$ in line 3. Line 4: we start the iteration at 2 as the definition of NN only makes sense if there is at least 2 time series. We then proceed with some initializations (lines 5–7), including for the cache associated with S (line 7). We are then ready to find (a) the NN of S within $\mathcal{T}'$, and (b) update the NN for all $T \in \mathcal{T}'$ now that S has been added. We will do this operation for all w in descending order, starting with the largest value $L - 1$ (line 8). Note that to only assess a subset of all possible L values for w, one only need to modify this line; our only requirement is for the values to be taken in descending order.

Line 9: we start by checking if we already have a NN for S from previous (i.e. larger) windows. Note that $\mathrm{NNs}[s][w]$ here comes compulsorily from DTW$_{w',w'>w}(S,\_)$ for which its path is still valid. We then already have the NN of S and only need to check whether S has also become a NN for other time series in $\mathcal{T}'$ (lines 10–14). To this end, we get the previously calculated NN for such $T \in \mathcal{T}'$ (line 11), which we will use as the threshold of distance that we have to 'beat' (i.e. be smaller than) for S to become the NN of T. On line 12 we then call our LAZYASSESSNN function to assess if S has actually become the NN of T. If LAZYASSESSNN exits with `pruned`, it then means that S is not the NN of T, and thus that the previous NN of T is still valid, hence nothing has to be done. LAZYASSESSNN only exits with something else than `pruned` if DTW$_w(S,T) <$ toBeat, which means that S has indeed become the NN of T; we update this information on line 14.

The `else` case starting on line 15 is if we didn't already have the NN of S from a previous window. We will then analyse all couple $(S,T)_{T\in\mathcal{T}'}$ and perform the NN update for S and T simultaneously. At this stage, it is possible that we will already have – from previous windows – some information about which $S \in \mathcal{T}'$ might be a better candidate to be the NN of S. This information is stored in the cache $\mathcal{C}$, which may contains different types of lower bounds. The number of calculations will be minimized if the very first T is actually the NN of S, because it will give the tightest possible pruning threshold first. This is why we should first examine the time series that have highest potential to become the NN of S and order the time series.[1]

At line 17, we obtain the distance threshold from $\mathrm{NNs}[s][w]$; the first time, we will have $\mathrm{NNs}[s][w] = \varnothing$ which is associated with a value of $+\infty$, there will thus be computation of DTW$_w(S,T)$, which will later on be stored in $\mathrm{NNs}[s][w]$ (line 20). From the second T, $\mathrm{NNs}[s][w]$`.distance` stores the distance to the so-far NN of S; we use LAZYASSESSNN to prune candidates or replace the current one if it is better (lines 19–20). We then proceed by checking if S is the NN of T on lines 21–24 (same as lines 11–14). Finally on line 25, having

[1]Ranking LB_KEOGH lower bounds has been previously studied in [1]. Here, we however have different types of lower bounds to interlace. LB_KIM often has a smaller value than LB_KEOGH simply because it only looks at 4 elements of series (vs L for LB_KEOGH). In addition, because DTW$_{w',w'>w}$ has tried to align S and T, it will probably represent a better estimate of DTW$_w$ than LB_KEOGH. To reflect this, we rank the time series in $\mathcal{T}'$ using LB_KIM $/4$, LB_KEOGH $/L$, $0.8 \cdot$ DTW $/L$ (the 0.8 factor is used to push DTW forward when close to the LB_KEOGH values).

processed all $T \in \mathcal{T}'$, $\text{NNs}[s][w]$ contains the actual NN of S at w, and we propagate this information for all $w', w' < w$ for which the warping path is also valid. Note that the cache $\mathcal{C}$ is never reused once the row $\text{NNs}[s]$ is computed, which makes it possible for our algorithm to have $\Theta(N)$ memory complexity.

4.4 Empirical Evaluation

This section describes the experiments that evaluate our FASTWWSEARCH method. To facilitate others to build on our work, as well as to ensure reproducibility, we have made our code available open-source at https://github.com/ChangWeiTan/FastWWSearch and the full raw results at http://bit.ly/SDM18. We compare FASTWWSEARCH's ability to learn the warping window compared to the state of the art:

- **LB_KEOGH** [27]: It searches for the best warping window as described in Algorithm 9 using LB_KEOGH as LB.
- **UCR_SUITE** [32]: It is the state of the art for fast NN-DTW and uses cascading lower bounds to replace LB in Algorithm 9. Note that we omitted the optimisation on z-normalisation for UCR_SUITE as all the datasets [6] are already z-normalised.
- **LB_KEOGH–PRUNEDDTW**: The PRUNEDDTW algorithm [93] was introduced to speed up the calculation of DTW using upper bounds (instead of lower bounds). It assesses warping windows in ascending order and uses the results for a smaller w as the upper bound for the larger w. Note that we actually improve here on the original paper [93] by adding LB_KEOGH to the search mechanism.
- **UCR_SUITE–PRUNEDDTW:** To make the comparison as fair as possible, we propose to combine the power of UCR_SUITE's lower bounding and of PRUNEDDTW'upper bounding. Again, this method is an improvement on both methods.

We performed our experiments using all of the 85 freely available benchmark UCR time series datasets [6] and use the original train/test split from [6]. Note that these datasets are provided z-normalized but our method is directly applicable to unnormalized series. We

Algorithm 12: FASTFILLNNTABLE($\mathcal{T}$)

```
Input: $\mathcal{T}$ the set of time series
Result: $\mathrm{NNs}[N][L]$ the nearest neighbors table
1  Define LANN as LAZYASSESSNN
2  $\mathrm{NNs}$.fillAll($\_, +\infty$)
3  $\mathcal{T}' \leftarrow \emptyset$
4  for $s \leftarrow 2$ to $N$ do
       // We want to update $\mathrm{NNs}$ wrt adding $S$
5      $S \leftarrow \mathcal{T}_s$
6      $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\mathcal{T}_{s-1}\}$
7      foreach $T \in \mathcal{T}'$ do $\mathcal{C}_{S,T} \leftarrow \emptyset$
8      for $w \leftarrow L-1$ down to $0$ do
           // If we already have NN of $S$ for $w$
9          if $\mathrm{NNs}[s][w] \neq \emptyset$ then
               // Update table $\mathrm{NNs}[t][w]_{1 \leqslant t \leqslant s-1}$
10             for $t \leftarrow 1$ to $s-1$ do
11                 toBeat $\leftarrow \mathrm{NNs}[t][w]$.distance
12                 res $\leftarrow$ LANN($\mathcal{C}_{(S,T_t)}, w,$ toBeat$, S, T_t$)
13                 if res $\neq$ pruned then
14                     $\mathrm{NNs}[t][w] \leftarrow (S,$ res$)$
15                 end
16             end
17         else
               // Check $S$ against previous $T \in \mathcal{T}'$
18             foreach $T \in \mathcal{T}'$ in asc. order using $\mathcal{C}$ do
19                 toBeat $\leftarrow \mathrm{NNs}[s][w]$.distance
20                 res $\leftarrow LANN(\mathcal{C}_{(S,T)}, w,$ toBeat$, S, T)$
21                 if res $\neq$ pruned then
22                     $\mathrm{NNs}[s][w] \leftarrow (T,$ res$)$
23                 end
                   // Update $\mathrm{NNs}[t][w]$ if needed
24                 toBeatT $\leftarrow \mathrm{NNs}[t][w]$.distance
25                 resT $\leftarrow LANN(\mathcal{C}_{(S,T)}, w,$ toBeatT$, S, T)$
26                 if resT $\neq$ pruned then
27                     $\mathrm{NNs}[t][w] \leftarrow (S,$ resT$)$
28                 end
29             end
               // Propagate NN for path validity
30             for $w' \in \mathrm{NNs}[s][w]$.valid do $\mathrm{NNs}[s][w'] \leftarrow \mathrm{NNs}[s][w]$
31         end
32     end
33 end
```

perform an exhaustive search for all methods; as mentioned in Section 4.3.2, all methods are directly applicable to any subset of $[\![0, L-1]\!]$ that one might want to use instead of the full set. All methods have linear memory complexity, so we were able to conduct all experiments on a small machine (64-bit Linux with AMD Opteron 63xx Class CPU @1.80GHz and 6GB RAM). As the ordering of the time series in $\mathcal{T}$ affects all compared methods, the time to search for the best warping window, we report the average results over 10 runs for different reshuffles of $\mathcal{T}$. We report the full leave-one-out cross-validation running time.

All methods are exact: they all learn the same value of the warping window, and thus all return the same accuracy, which is why we focus on running time. A fail-safe check is presented in Appendix A and shows that all methods indeed learn the same warping window.

4.4.1 Speed-up

Figure 4.4 shows the scatter plot of learning time for our FastWWSearch method (x-axis) vs all competitors (y-axis). This is a clear-cut and significant result: our method is faster than the state of the art for all datasets (above the line means that our method is faster). There is also a slight upwards trend: as the task becomes more and more complicated, it seems that so is the improvement of our FastWWSearch method over the state of the art. For easy task requiring less than 10 seconds with the state of the art, FastWWSearch gains one order of magnitude and performs in less than 1 seconds. This makes sense and this is not where the gain is the most interesting. However, as the task becomes harder, so is the advantage of our method over others getting more important. This even reaches up to 3 orders of magnitude speed-up for very demanding tasks. For dataset `HandOutlines` for instance, the fastest state-of-the-art method requires 100 days ($9 \cdot 10^6$ s), while FastWWSearch only needs 2.5 hours ($9 \cdot 10^3$s).

It is also interesting to summarize the results depending on size of the data and length of the series. We calculated the average speed-up for datasets with smaller training size ($N \leqslant 200$) and larger training size ($N > 200$). The average speed-up for smaller datasets was of 106x, while it was of 184x for larger ones. Regarding length, we calculated the average speed-up for datasets with shorter series ($L \leqslant 300$) and longer ones ($L > 300$). The average speed-up for datasets with shorter series was of 67x, while it was of 250x for longer ones. These results confirm that our algorithm is tackling both the N and L terms.

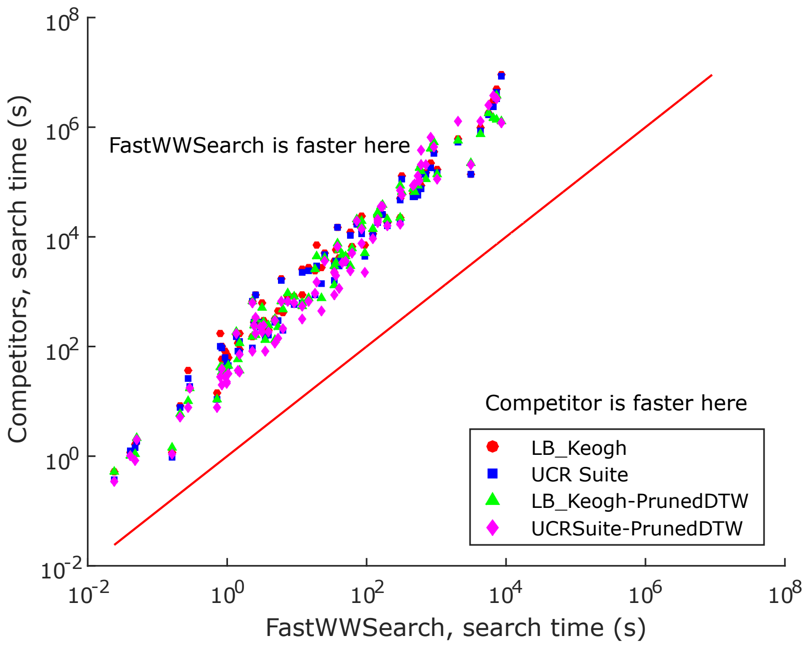


Figure 4.4: Average 10 runs results on the benchmark datasets (better seen in color)

4.4.2 Scalability to 100,000 Time Series

It is now interesting to comment again on Figure 4.1 that was presented in the introduction. This dataset corresponds to 100,000 time series, associated to 100,000 'pixels' observed over time (over a series of satellite images) (more details in [10]). In this task, the aim is to establish the land-use of a geographic area based on its radiometric evolution over a series of satellite images. Time series are required because, for instance, one needs the temporal dynamic to distinguish between types of crops (when they grow and are harvested, how fast they grow, etc). Note that we used a computer with 16GB memory for this experiment to be able to store the data.

This dataset is also particularly interesting because the time series are short with $L = 46$, which tends to isolate the influence of N on the scalability. We can see in Figure 4.1 that when $N = 100$, FASTWWSEARCH makes it possible to gain 1.5 orders of magnitude in running time over UCR_SUITE and 2 orders over the other state-of-the-art methods; and those gains seem very stable when N grows. For $N = 100,000$, FASTWWSEARCH only requires 6 hours to complete, while the fastest state-of-the-art method – in this case UCR_SUITE – requires 7 days (and more than 18 days for the others).

4.4.3 Incorporating PrunedDTW Within FastWWSearch

It is interesting to examine if PrunedDTW could provide further improvements to our method. As the PrunedDTW algorithm [93] is able to speed up DTW computations, it is interesting to study if its incorporation into our algorithm could bring further benefits. Our method requires the different windows to be tested in descending order, in order to reuse previously calculated results as lower bounds to current ones. This is incompatible with a PrunedDTW search, which requires the windows to be assessed in ascending order, to use previous results as upper bounds to the next ones [93]. However, we could still use PrunedDTW whenever we have to calculate DTW, and use the Euclidean distance as a general upper bound [93]. This is how we incorporate PrunedDTW into FastWWSearch.

Figure 4.5 compares our original FastWWSearch with a version incorporating PrunedDTW. The results show that both methods have similar running times, with the addition of PrunedDTW making it possible to gain some speed-up for low-runtime datasets (i.e. datasets that are either small or have short time series or both). Using FastWWSearch vanilla seems to be even faster for high-runtime datasets. Overall, for approximately 55% of the UCR archive, FastWWSearch vanilla is faster than having added PrunedDTW. This is because for high-complexity datasets, it seems that the added pruning power doesn't outweigh the additional computations of the upper bound that PrunedDTW requires. Overall, the take-home message here is that our method is totally compatible with PrunedDTW and that its incorporation is left at the discretion of the data practitioner, depending on their application.

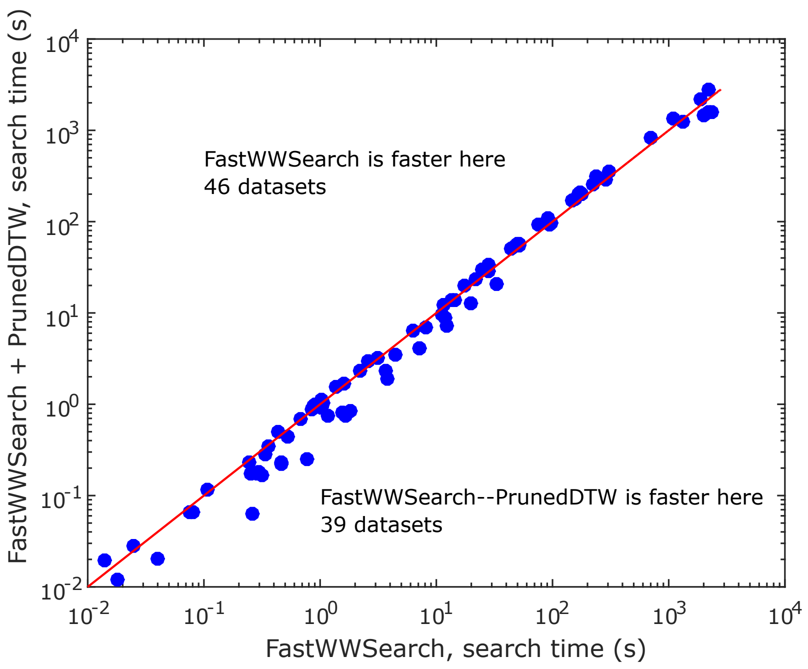


Figure 4.5: Comparison of FASTWWSEARCH with the PRUNEDDTW implementation on the benchmark datasets [6]

4.5 Conclusion

In this work, we proposed FASTWWSEARCH: a novel algorithm and underlying theory to efficiently learn the warping window for Dynamic Time Warping. Our experiments show that it is one to two orders of magnitude faster than the state of the art. This result is important both for the use of NN-DTW and also for incorporation into the state-of-the-art classifiers EE and COTE.

4.6 Acknowledgement

This work was supported by the Australian Research Council under grant DE170100037. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-16-1-4023. The authors would like to thank Professor Eamonn Keogh, Hoang Anh Dau and anonymous reviewers for their comments on this paper.

Chapter 5

Time Series Classification with Fast Ensembles of Elastic Distances

In recent years, many new ensemble-based time series classification (TSC) algorithms have been proposed. Each of them is significantly more accurate than their predecessors. The Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) is currently the most accurate TSC algorithm when assessed on the UCR repository. It is a meta-ensemble of 5 state-of-the-art ensemble-based classifiers. The time complexity of HIVE-COTE – particularly for training – is prohibitive for most datasets. There is thus a critical need to speed up the classifiers that compose HIVE-COTE. This paper focuses on speeding up one of its components: *Ensembles of Elastic Distances* (EE), which is the classifier that leverages on the decades of research into the development of time-dedicated measures. Training EE can be prohibitive for many datasets. For example, it takes 2-3 weeks on the `ElectricDevices` dataset with 9,000 instances. This is because EE needs to cross-validate the hyper-parameters used for the 11 similarity measures it encompasses. In this work, *Fast Ensembles of Elastic Distances* is proposed to train EE faster. There are two versions to it. The exact version makes it possible to train EE 10 times faster. The approximate version is 40 times faster than EE without significantly impacting the classification accuracy. This translates to being able to train EE on `ElectricDevices` in 13 hours.

5.1 Introduction

Time series data are growing at an unprecedented rate. One of the largest publicly available training datasets currently holds 1,000,000 satellite image time series instances [1]. These data describe the evolution of an area on Earth and are used to create land-cover maps. Data of this scale are just the tip of the iceberg. Typically, the creation of land-cover maps require at least 100 million time series [8, 9]. The computational demands of learning from these huge amounts of data challenges state-of-the-art techniques [1, 3]. For instance, searching through long ECG queries with 0.3 trillion data-points using the Nearest Neighbour classifier with Euclidean distance (NN-ED) without any optimisation took a week [32]. Note that Euclidean distance is the fastest to compute out of the standard similarity measures, as it is linear with the length of the time series. At a smaller scale, processing 45 minutes of speech data using the Dynamic Time Warping (DTW) distance took 3 hours on a single core machine [121].

Time series classification (TSC) is an important tool for these applications. Currently the most accurate TSC algorithm is the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [15]. HIVE-COTE is an ensemble of five groups of classifiers that uses a hierarchical voting scheme to weight the predictions from each group of classifiers [15]. The intuition is that, combining classifiers from different domains should perform better than using classifiers from a single domain [21]. The superiority in classification accuracy requires learning of the parameters at training time [16].

Unfortunately, training these ensemble-based classifiers is very computationally demanding and is infeasible for large datasets. Two of the core classifiers in HIVE-COTE with the highest training time are the Shapelet Ensembles (SE) [15] and the Ensembles of Elastic Distances (EE) [17]. SE is an ensemble that combines 8 base learners and uses the transformed time series data from the Shapelet Transform (ST) algorithm [60]. Given a training set of N time series with length L, the ST algorithm has a training time complexity of $O(N^2 \cdot L^4)$.

EE is an ensemble of NN classifiers with 11 different time series distance measures. State-of-the-art learning of the parameters for all 11 classifiers is done through leave-one-out cross validation (LOO-CV). It has a training complexity of $O(M \cdot N^2 \cdot L^2)$, where $M = 100$ is the number of parameter values to learn for each classifier in EE [15, 23]. The high training complexity is because the learning algorithm needs to compare each of the time series in the training set of size N with $N - 1$ other instances and each comparison typically requires expensive $O(L^2)$ operations. This process is then repeated for all M parameter values, which is very inefficient and impractical for time series datasets with large N and long L.

In this work, we propose the Fast Ensembles of Elastic Distances (FASTEE) to reduce EE training time. FASTEE extends recent work on speeding up the training time for the NN-DTW algorithm [3] to other distance measures. There are 2 versions of FASTEE. The exact version trains FASTEE using the full LOO-CV. The approximate version uses approximate LOO-CV. FASTEE minimizes the number of distances that need to be calculated by determining for each value calculated the range of parameter values for which that distance value applies, thus saving the majority of distance calculations undertaken by the standard approach. It also uses lower bounds on the distance measures to speed up each of the NN classifiers, where new lower bounds are proposed for distances for which no previous bounds have been derived.

We show the significance of our FASTEE algorithms in Figure 5.1, which compares the training time of EE to our FASTEE algorithms as a function of training set size. To generate this plot, we sampled the `ElectricDevices` dataset – the largest dataset from the UCR benchmark time series archive [6] at different sizes and report the average training time. The red line shows that training EE on this dataset with merely 9,000 instances takes 17 days. Our exact FASTEE, in blue, reduces this time to 2 days, while the approximate FASTEE, in green, reduces it to 13 hours. The exact version is 10 times faster than EE, while learning the exact same classifier. The approximate version is 40 times faster, but may slightly compromise classification accuracy.

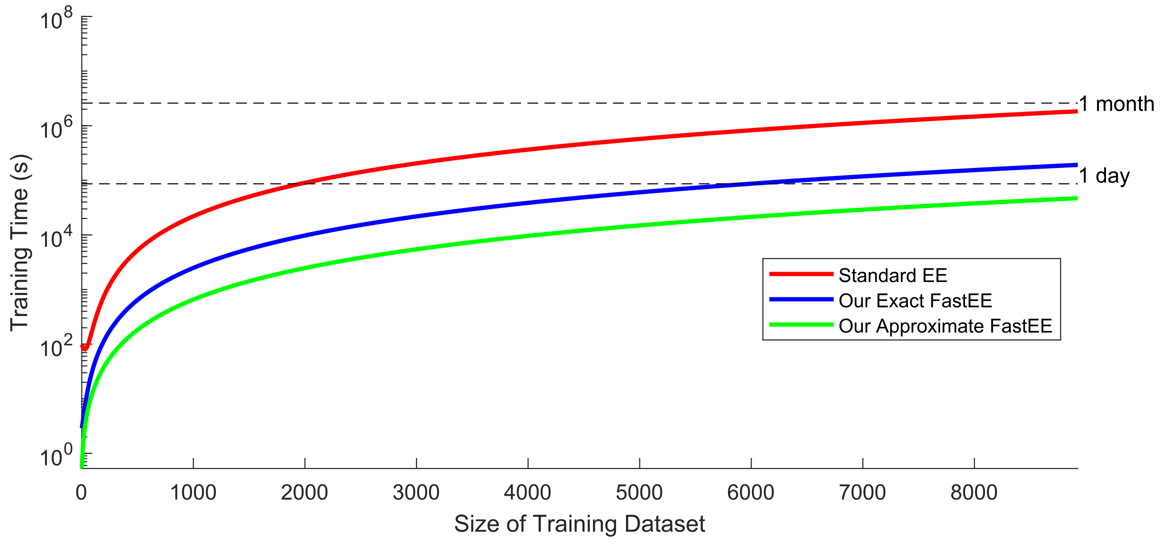


Figure 5.1: Training time of EE (17 days) and our proposed technique FastEE (2 days) on the `ElectricDevices` dataset at different sizes.

This chapter is organised as follows. We provide the necessary background to understand our work and review key related work in TSC in Section 5.2. As lower bounding a distance measure has shown numerous successes in speeding up NN classifiers [27, 31, 32], we believe that it can also speed up EE. Section 5.3 describes the lower bounds required to speed up EE. We also propose new lower bounds for distance measures for which lower bounds have not previously been derived. Then in Section 5.4, we introduce FastEE to speed up the training time for EE. We evaluate the performance of FastEE in Section 5.5. Finally, we conclude our chapter and provide some future direction in Section 5.6.

5.2 Background and Related Work

Most of the background knowledge have been presented in Chapter 2. This section provides the necessary background information to understand our proposed method and describes some related work in speeding up time series classification (TSC), specifically the nearest neighbour (NN) type classifiers. For simplicity, we assume that all the time series are of the same length L. However, it is trivial to generalize the algorithm to different lengths.

5.2.1 Ensembles of Elastic Distances

Similarity in the time domain has been a major focus for a large body of TSC research [1, 10, 16, 17, 22, 23, 27, 93]. The Ensembles of Elastic Distances (EE) is the most accurate TSC classifier in the time domain [17, 23]. EE is a meta classifier that consists of 11 NN classifiers, each of which uses a different time series distance measure [17] (see Section 2.2.9 for more details). A NN classifier searches for the nearest neighbour within the training set and labels the query time series with the label of the nearest neighbour. Two time series are compared using a distance measure such as Euclidean distance or Dynamic Time Warping (DTW), that tries to achieve the optimal alignment between the two time series. Many distance measures are proposed for TSC in the last decade [23, 45, 88, 89, 90, 91]. When tested on the UCR benchmark archive, they are not significantly different from each other in terms of classification accuracy [17]. An ensemble (EE) generalises this so that the contributions from all the distance measures are considered, which significantly improves the classification accuracy [17].

As is typical for supervised classifiers, EE needs to be trained before performing any classification tasks. One of the key aspects of learning for EE is to find values for the parameters that affect the behaviour of the distance measures [17]. For instance, the choice of warping window (parameter) for one of the distance measures – Dynamic Time Warping (DTW) significantly affects the error rate of NN-DTW, as shown in Figure 5.2. EE learns the values of these parameters by trying all of a predefined set of potential values and selecting the one that performs best using leave-one-out cross validation (LOO-CV) on the training set [17]. Since EE is composed of NN type classifiers, this task then boils down to finding the nearest neighbour of each time series inside the training dataset for each of M parameter values. Algorithm 2 in Section 2.2.9 describes the process of training EE.

5.2.2 Elastic Distance Measures

Since EE consists of NN type classifiers, the ultimate aim of this work is to minimise the training time of NN type classifiers. The NN type classifiers are each paired with a distance

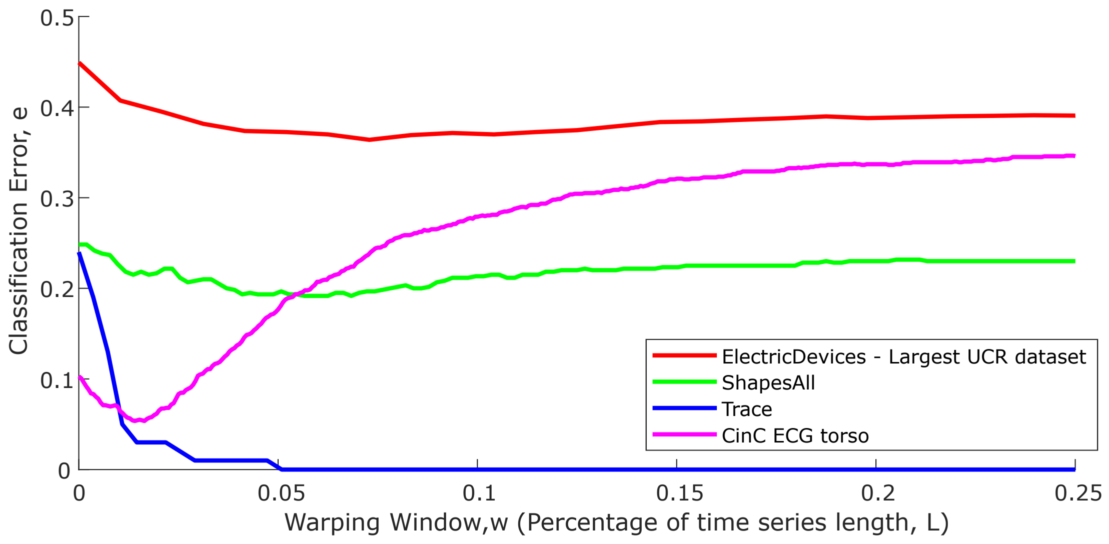


Figure 5.2: Classification error for NN-DTW on some datasets from the UCR benchmark archive [6] at various warping window w

measure, also known as a similarity measure, that returns a distance, or similarity, score for a pair of time series. Thus it is important to first understand the principles of each distance measure. There are 7 different distance measures that are commonly used in the literature [17]. Together with their variants, they form with 11 distance measures used in EE [17]. This section briefly describes the different measures in order to understand our work in this chapter. We refer interested readers to Section 2.2 and the paper [17] for more technical details.

An elastic distance measure E transforms a time series to better match and align with another time series. All distance measures in EE are *elastic* other than Euclidean distance. Most of the elastic distance measures have $O(L^2)$ complexity to calculate. They can all be solved with dynamic programming using a $L \times L$ cost matrix $\mathcal{D}$. Thus, we say that E has an *alignment* path $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_K\}$ along the cost matrix $\mathcal{D}$ that aligns the two time series S and T. Each of the element $\mathcal{A}_k = (i, j)$ is a link indicating $S(i)$ is aligned to $T(j)$. Equation 5.1 shows a general equation for elastic distance measures, where $\mathcal{A}_k^1$ indicates the first element in $\mathcal{A}_k$, $\mathcal{A}_k^2$ the second, $\texttt{cost}_{\mathcal{P}}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$, represents the cost of aligning the two points $S(\mathcal{A}_k^1)$ and $T(\mathcal{A}_k^2)$, given a parameter $\mathcal{P}$ and is explained in Section 2.2.

$$E(S,T) = \sum_{k=1}^{K} \texttt{cost}_{\mathcal{P}}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2)) \tag{5.1}$$

Elastic Distance Measures	**Time Complexity**	**Path Constraint**	**Alignment Penalty**	**Threshold**
Euclidean Distance	$O(L)$	-	-	-
Dynamic Time Warping	$O(L^2)$	-	-	-
Constrained DTW	$O(w \cdot L)$	w	-	-
Derivative DTW	$O(L^2)$	-	-	-
Constrained DDTW	$O(w \cdot L)$	w	-	-
Weighted DTW	$O(L^2)$	-	g	-
Weighted DDTW	$O(L^2)$	-	g	-
Longest Common Subsequence	$O(\delta \cdot L)$	δ	-	ε
Edit Distance with Real Penalty	$O(\texttt{bandsize} \cdot L)$	`bandsize`	g	-
Move-Split-Merge Distance	$O(L^2)$	-	`c`	-
Time Warp Edit Distance	$O(L^2)$	v	λ	-

Table 5.1: Elastic distance measures in EE with their time complexity and parameters

The elastic distance measures are usually parametrised by one or two parameters that need to be learned at training time. Table 5.1 outlines the respective time complexity and parameters for each of the elastic distance measures. Overall, the parameters for elastic distance measures can be categorised into three types. First, is the alignment constraint parameter on the alignment path such as the warping window in DTW. Second, is a penalty cost to the alignment in case of misalignment and lastly, a threshold parameter which determines if two points are the same. Most elastic distance measures are parametrised by one or two of these forms of parameters.

Algorithm 13 presents a general algorithm to fully compute any given elastic distance measure E (without a global alignment constraint) given their threshold or alignment penalty parameters, denoted by a single variable $\mathcal{P}$ and the cost of aligning two points. For the case where the warping path is constrained, one only need to modify the $start$ and end variables. For a constraint variable $\texttt{C}$, we replace the respective $start$ and end variables: $end \leftarrow \texttt{C}$ in Line 3, $start \leftarrow \max(2, i - \texttt{C})$ in Line 12 and $end \leftarrow \min(L, i + \texttt{C})$ in Line 13. Note that $\texttt{C} = r$ for DTW$_r$ and DDTW$_r$; $\texttt{C} = \Delta$ for LCSS; $\texttt{C} =$ bandsize for ERP.

5.2.3 Related Work

The accuracy of the EE classifier, comes at a cost of polynomial time complexity, which is prohibitive for large datasets. As shown in the previous section, elastic distance measures generally have polynomial $O(L^2)$ time complexity. A single classification of a query using the

Algorithm 13: Full Elastic Distance, $E(S, T, \mathcal{P}, \mathcal{D})$

```
Input: $Q$: Query time series
Input: $C$: Candidate time series
Input: $\mathcal{P}$: Parameters for elastic distance measure, $E$
Input: $\mathcal{D}$: Cost matrix for elastic distance measure, $E$
Output: $d$: Value of the elastic distance measure
$L \leftarrow S.\texttt{length}$
Let $\mathcal{D}$ be an $L \times L$ matrix initialized to $\infty$
$end \leftarrow L$
$\mathcal{D}(1,1) \leftarrow \texttt{cost}_{\mathcal{P}}(S(1), T(1)$
for $i \leftarrow 2$ to $end$ do                                   // first column
    $\mathcal{D}(i,1) \leftarrow \texttt{cost}_{\mathcal{P}}(S(i), T(1))$
end
for $j \leftarrow 2$ to $end$ do                                   // first row
    $\mathcal{D}(1,j) \leftarrow \texttt{cost}_{\mathcal{P}}(S(1), T(j))$
end
for $i \leftarrow 2$ to $L$ do                                     // the rest
    $start \leftarrow 2$
    $end \leftarrow L$
    for $j \leftarrow start$ to $end$ do
        $\mathcal{D}(i,j) \leftarrow \texttt{cost}_{\mathcal{P}}(S(i), T(j))$
    end
end
if LCSS then
    return $1 - \mathcal{D}(L,L)/L$
else
    return $\mathcal{D}(L,L)$
end
```

NN classifier requires $O(N \cdot L^2)$ operations, as the the algorithm needs to compare the query time series with all the N training instances. On the other hand, as indicated in Algorithm 2, with M parameter values to select between, a typical training time for a single NN classifier in EE requires $O(M \cdot N^2 \cdot L^2)$ operations. This is because for each of the N instances in the training set $\mathcal{T}$, the algorithm needs to scan through all $N - 1$ examples to find its nearest neighbour and repeat for all M parameter values. This is infeasible when L is long and N is large as illustrated in Figure 5.1. An exhaustive search of all M parameters is time consuming, thus many applications use a subset of parameter values by setting $M = 100$. However as we will show in Section 5.5, training with $M = 100$ is still inefficient and slow.

A great amount of research has been done to speed up NN type classifiers, in particular the NN-DTW classifier [1, 26, 27, 28, 31, 32, 93]. A common way is to use efficient lower bound functions with $O(1)$ to $O(L)$ complexity to minimise the expensive $O(L^2)$ distance computation and to reduce the search space [27, 28, 30, 31, 32]. Different lower bounds can be cascaded to create tighter lower bounds that are more effective at pruning unpromising candidates [32]. It may be possible to determine lower bounds on the final distance during the dynamic programming process. Thus, the computation of an elastic distance measure can also be early abandoned [32]. The idea is to abandon the distance computation as soon as the cumulative distance in the cost matrix $\mathcal{D}$ is greater than the distance to the nearest neighbour [32]. Cells in the cost matrix $\mathcal{D}_{\text{DTW}}$ that are guaranteed to not be part of the DTW warping path can be skipped as well [93].

Contract algorithms are also popular to speed up a classification algorithm in terms of training and classification time without compromising on the classification accuracy [1, 26, 103]. A recent work proposed c-RISE [103] that gives a contract training time for the Random Interval Spectral Ensemble (RISE) – a component of HIVE-COTE [15]. Indexing techniques such as building a hierarchical K-means tree when combined with contract algorithms [1] are effective in reducing the classification time. Moreover, we can also classify a time series by comparing it to the average time series in each of the classes in the training set [26]. This significantly reduces the classification time and can improve the classification accuracy [26].

Since most elastic distance measures are very similar, these techniques developed for NN-DTW can be extended to other elastic distance measures to speed up this process. As will be discussed in Section 5.4, FastEE leverages off a recent work that utilises some of these techniques.

5.2.4 Learning the Parameters of an Elastic Distance Efficiently

Most of the elastic distance measures require learning the parameter from a set of M parameter values [16, 23]. Typically, the training time for a single NN classifier in EE requires $O(M \cdot N^2 \cdot L^2)$ operations and is infeasible for large N. Although there has been much research into speeding up the NN classifier (Section-5.2.3), they are still not efficient in dealing with hundreds of parameter values. For instance, our experiments in Section 5.5 show that there is no significant improvement in training time by using just the lower bounds. Further-

	Nearest neighbour at different parameters				
	0	1	$\cdots$	$M-1$	M
T_1	$T_{24}(2.57)$	$T_{55}(0.98)$	$\cdots$	$T_{55}(0.98)$	$T_{55}(0.98)$
$\vdots$			$\vdots$		
T_N	$T_{60}(4.04)$	$T_{47}(1.61)$	$\cdots$	$T_{47}(1.61)$	$T_{47}(1.61)$

Table 5.2: Table of NNs for each m. A cell $(i, m) = T_k(dist)$ means T_i has T_k as its NN for parameter m with distance $dist$.

more, as demonstrated in [3], learning the exact best warping window (parameter) for DTW requires the enumeration of all possible windows ($M = L$) which is a laborious process [3]. This process can be seen as searching for the nearest neighbour of all N time series for all windows, creating a $(N \times M)$ NNs table as shown in Table 5.2 [3]. Filling this table naively requires $O(N^2 \cdot L^3)$ operations. Thus, it is common to explore just a subset of these windows, giving an approximate best warping window. Usually at least 100 windows in the range of 0 to L are considered [23]. However this is still not efficient, as the complexity of $N^2 \cdot L^2$ takes over very quickly for large N and long L.

The *Fast Warping Window Search* (FastWWS) is an exact and efficient algorithm to learn the best warping window for DTW that is at least 2 orders of magnitude faster than the state of the art [3]. The algorithm is based on the observation that many distance computations are redundant in the standard approach to learning the best warping window [3]. It takes advantage of the relationship between DTW and its warping window [3]. There are three main properties that form the basis of FastWWS [3]. First, *a* DTW *warping path can be valid for several windows*. When unconstrained, DTW finds the optimum path that goes through the cost matrix $\mathcal{D}$. This optimum path has a maximum width $w^\star$ from the diagonal. If the window w is larger than $w^\star$, then the path will not change for any smaller window $w^\star \leq w' \leq w$ and thus will return the same distance, as illustrated in Figure 4.3. Warping windows whose DTW distances are predetermined need not be computed. Second, DTW *is monotone with the warping window*. Lastly, *The Keogh lower bound (lower bound of* DTW*) is monotone with warping window* as well. Section 2.4 explains the lower bounds in more detail. These three properties allow the searching for the best warping window to start with the largest window L and proceed through ever smaller windows, skipping the computation of many of the DTW distances. We refer the interested reader to the paper [3] or Chapter

4 for a detailed explanation of the algorithm. With the aim of speeding up EE, this work generalises these properties and extends FASTWWS to other elastic distance measures.

5.3 Proposed Lower Bounds for Elastic Distances

Lower bounding has shown a lot of success in speeding up the NN classifier especially the NN-DTW classifier [1, 27, 32]. If a sequential search is performed through potential nearest neighbours for series S, if a lower bound on the distance to T exceeds the distance to the nearest neighbour found so far, then T can be discarded as a potential nearest neighbour without the need to calculate its distance. This speeds up NN type classifiers by minimising the number of expensive $O(L^2)$ distance computations. Typically an effective lower bound has cheap $O(1)$ to $O(L)$ complexity. EE is an ensemble of 11 NN classifiers with different elastic distance measures, thus it is possible to speed up EE using lower bounds. Section 2.4 describes the existing lower bounds that are used in this work.

To the best of our knowledge, no prior lower bounds have been derived for WDTW, MSM or TWED. In this section, we present new lower bounds for these measures. Note that some of these lower bounds may not be tight. Nonetheless, they are sufficient to provide reasonable speed ups. With the addition of these lower bounds, we have useable lower bounds for all the elastic measures in EE. We do not use a lower bound for Euclidean distance because its complexity is linear with respect to series length and hence it is efficient

5.3.1 WDTW Lower Bound

We define the lower bound for WDTW (LB_WDTW) in Equation 5.2 using similar intuition as LB_KEOGH. First, we build the envelope time series for T. Since there are no alignment constraints on WDTW, we have the upper envelope $UE = \max(T)$ and the lower envelope $LE = \min(T)$. Then LB_WDTW distance is computed using Equation 5.2 by multiplying the sum with the minimum weight penalty $\mathtt{w}_0$.

$$\text{LB\_WDTW}(S,T) = \sqrt{\mathtt{w}_0 \sum_{i=1}^{L} \begin{cases} (S(i) - \max(T))^2 & \text{if } S(i) > \max(T) \\ (S(i) - \min(T))^2 & \text{if } S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases}} \tag{5.2}$$

Theorem 5. *For any two time series S and T of length L, and an alignment path $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_K\}$, where $\mathcal{A}_k = (i,j)$ indicates S_i is aligned to T_j, the following inequality holds:* LB_WDTW$(S,T) \leq$ WDTW(S,T)

Proof. We can rewrite the equation of WDTW using $\mathcal{A}$ and the weights from Equation 2.4 into the following

$$\text{WDTW}(S,T) = \sqrt{\sum_{k=1}^{K} \mathtt{w}_{|\mathcal{A}_k^1 - \mathcal{A}_k^2|} (S(\mathcal{A}_k^1) - T(\mathcal{A}_k^2))^2}$$

where $\mathcal{A}_k^1 = i, \mathcal{A}_k^2 = j$ and we wish to proof the following

$$\mathtt{w}_0 \sum_{i=1}^{L} \begin{cases} (S(i) - \max(T))^2 & \text{if } S(i) > \max(T) \\ (S(i) - \min(T))^2 & \text{if } S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases} \leq \sum_{k=1}^{K} \mathtt{w}_{|\mathcal{A}_k^1 - \mathcal{A}_k^2|} (S(\mathcal{A}_k^1) - T(\mathcal{A}_k^2))^2$$

Note that both sides are squared as the terms under the square root are both positive. We know that $L \leq K$, so we can match every term on the left hand side (LHS) with a term on the right hand side (RHS) giving $K - L$ terms unmatched.

$$\mathtt{w}_0 \sum_{i=1}^{L} \begin{cases} (S(i) - \max(T))^2 & \text{if } S(i) > \max(T) \\ (S(i) - \min(T))^2 & \text{if } S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases} \leq \begin{array}{l} \sum_{k \in matched} \mathtt{w}_{|\mathcal{A}_k^1 - \mathcal{A}_k^2|} (S(\mathcal{A}_k^1) - T(\mathcal{A}_k^2))^2 + \\ \sum_{k \in unmatched} \mathtt{w}_{|\mathcal{A}_k^1 - \mathcal{A}_k^2|} (S(\mathcal{A}_k^1) - T(\mathcal{A}_k^2))^2 \end{array}$$

Let us consider the relationship between the matched terms on the RHS and the LHS terms. There are three cases to be considered on the LHS of the inequality. We start with the first one $S(i) > \max(T)$. From Equation 2.4, $\mathtt{w}_0$ is the minimum weight, so $\mathtt{w}_0 \leq \mathtt{w}_{|i-j|}$.

Since $\max(T) \geq T(j)$ and if $S(i) > \max(T)$, then $(S(i) - \max(T))^2 \leq (S(i) - T(j))^2$. Thus $\mathtt{w}_0(S(i) - \max(T))^2 \leq \mathtt{w}_{|j-i|}(S(i) - T(j))^2$. Similarly if $S(i) < \min(T)$, and $\min(T) \leq T(j)$, then $(S(i) - \min(T))^2 \leq (S(i) - T(j))^2$. Since $(S(i) - T(j))^2$ is non-negative, the third case yields $0 \leq (S(i) - T(j))^2$.

If all the matched terms are larger than the LHS terms, then the unmatched terms will need to be negative for the inequality to be false. Fortunately, it is impossible for the unmatched terms to be negative since $\mathtt{w}_{|i-j|} > 0$ (from Equation 2.4) and the squared terms can never be negative. Therefore our inequality holds and LB_WDTW$(S, T) \leq$ WDTW(S, T). $\square$

5.3.2 MSM lower bound

We define the lower bound for MSM (LB_MSM) in Equation 5.3. The first term of LB_MSM is $|S(1) - T(1)|$ and is extracted directly from MSM [90]. Following Equation 2.7, if $S(i-1) \geq S(i) > \max(T)$, the lower bound adds the minimum of $|S(i) - \max(T)|$ and $\mathtt{c}$. Similarly if $S(i-1) \leq S(i) < \min(T)$, the lower bound adds the minimum of $|S(i) - \min(T)|$ and $\mathtt{c}$.

$$\text{LB\_MSM}(S,T) = |S(1) - T(1)| + \sum_{i=2}^{L} \begin{cases} \min(|S(i) - \max(T)|, \mathtt{c}) & \text{if } S(i-1) \geq S(i) > \max(T) \\ \min(|S(i) - \min(T)|, \mathtt{c}) & \text{if } S(i-1) \leq S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Theorem 6. *For any two time series S and T of length L, and an alignment path $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_K\}$, where $\mathcal{A}_k = (i, j)$ indicates $S(i)$ is aligned to $T(j)$, the following inequality holds:* LB_MSM$(S, T) \leq$ MSM(S, T)

Proof. We can rewrite the equation of MSM from Equation 2.8 using $\mathcal{A}$ into the following:

$$\text{MSM}(S,T) = |S(1) - T(1)| + \sum_{k=2}^{K} \mathtt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$$

where $\mathcal{A}_k^1 = i, \mathcal{A}_k^2 = j$, $\mathtt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$ represents the cost of aligning $S(\mathcal{A}_k^1)$ to $T(\mathcal{A}_k^2)$ under MSM. From Equation 2.8, MSM is based on three different cost values.

$$\texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2)) = \begin{cases} |S(i) - T(j)| \\ \mathcal{C}(S(i), S(i-1), T(j)) \\ \mathcal{C}(T(j), S(i), T(j-1)) \end{cases}$$

where $\mathcal{C}(x, y, z)$ is defined in Equation 2.7. The cost is $\mathtt{c} + \min(|x-y|, |x-z|)$ if x is not between y and z otherwise the cost is $\mathtt{c}$. And we wish to proof the following

$$\sum_{i=2}^{L} \begin{cases} \min(|S(i) - \max(T)|, \mathtt{c}) & \text{if } S(i-1) \geq S(i) > \max(T) \\ \min(|S(i) - \min(T)|, \mathtt{c}) & \text{if } S(i-1) \leq S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases} \leq \sum_{k=2}^{K} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$$

The proof for the first term is trivial as they are equal on both sides. We know that $L \leq K$, so we can match every term on the LHS with a term on the RHS, giving $K-L$ terms unmatched.

$$\sum_{i=2}^{L} \begin{cases} \min(|S(i) - \max(T)|, \mathtt{c}) & \text{if } S(i-1) \geq S(i) > \max(T) \\ \min(|S(i) - \min(T)|, \mathtt{c}) & \text{if } S(i-1) \leq S(i) < \min(T) \\ 0 & \text{otherwise} \end{cases} \leq \begin{array}{l} \displaystyle\sum_{k \in matched} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2)) + \\ \displaystyle\sum_{k \in unmatched} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2)) \end{array}$$

Let us consider the relationship between the matched terms on the RHS and the LHS terms. We wish to proof that for each $i > 1$ term, all the 3 cases inside LB_MSM is less than or equal to the minimum of the cost functions.

For the first case $S(i-1) \geq S(i) > \max(T), \max(T) \geq T(j)$, and assuming $|S(i) - \max(T)| \leq \mathtt{c}$,

- $|S(i) - \max(T)| \leq |S(i) - T(j)|$ because $\max(T) \geq T(j)$, i.e. $T(j)$ is further from $S(i)$ than $\max(T)$.
- $|S(i) - \max(T)| \leq \mathcal{C}(S(i), S(i-1), T(j))$ because $|S(i) - \max(T)| \leq \mathtt{c}$, $\mathtt{c} \leq \mathcal{C}(S(i), S(i-1), T(j))$ and $S(i-1) \geq S(i) > \max(T) \therefore \mathcal{C}(S(i), S(i-1), T(j)) = \mathtt{c}$ from Equation 2.7.

- $|S(i) - \max(T)| \leq \mathcal{C}(T(j), S(i), T(j-1))$ because $|S(i) - \max(T)| \leq$ c and c $\leq$ $\mathcal{C}(T(j), S(i), T(j-1))$ from Equation 2.7.

Assuming if c $\leq |S(i) - \max(T)|$, then c $\leq |S(i) - \max(T)| \leq |S(i) - T(j)|$, c $\leq \mathcal{C}(S(i), S(i-1), T(j)$ and c $\leq \mathcal{C}(T(j), S(i), T(j-1))$ are still valid and thus the above proof holds.

For the second case $S(i-1) \leq S(i) \leq \min(T), \min(T) \leq T(j)$, and $|S(i) - \min(T)| \leq$ c,

- $|S(i) - \min(T)| \leq |S(i) - T(j)|$ because $\min(T) \leq T(j)$, i.e. $T(j)$ is further from $S(i)$ than $\min(T)$.
- $|S(i) - \min(T)| \leq \mathcal{C}(S(i), S(i-1), T(j))$ because $|S(i) - \min(T)| \leq$ c, c $\leq \mathcal{C}(S(i), S(i-1), T(j))$ and $S(i-1) \leq S(i) < \min(T) \therefore \mathcal{C}(S(i), S(i-1), T(j)) =$ c from Equation 2.7.
- $|S(i) - \min(T)| \leq \mathcal{C}(T(j), S(i), T(j-1))$ because $|S(i) - \min(T)| \leq$ c and c $\leq \mathcal{C}(T(j), S(i), T(j-1))$ from Equation 2.7.

Similarly, the above proof holds for the case where c $\leq |S(i) - \min(T)|$. The third case is always true because all of the cost functions are non-negative.

Since all the matched terms are larger than LHS, then the sum of the unmatched terms has to be negative for the inequality to be false. This is not possible because c ≥ 0 is non negative and the absolute differences in the cost values and Equation 2.7 can never be negative. Therefore LB_MSM$(S, T) \leq$ MSM(S, T) holds. □

5.3.3 TWED Lower Bound

TWED takes into account the differences in timestamps $t_i - t_{i-1}$ in the cost of aligning a time series pair. Since in this work, we only consider time series that are equally spaced and use the indexes of the time series points as the timestamps, we will always have $t_i - t_{i-1} = 1$. A lower bound for TWED was proposed in [91] for range query search. The lower bound down-samples the time series and computes the TWED distance of the down-sampled time series. In a worst case scenario, the down-sampled time series could be the full time series and the lower bound will be more expensive to compute than the full distance. Thus, we define a new lower bound for TWED (LB_TWED) in Equation 5.4. Note that this lower bound is also applicable for $t_i - t_{i-1} > 1$.

$$\text{LB\_TWED}(S,T) = \min \begin{cases} (S(1)-T(1))^2 \\ S(1)^2 + v + \lambda \\ T(1)^2 + v + \lambda \end{cases} +$$

$$\sum_{i=2}^{L} \begin{cases} \min(v, (S(i) - \max(\max(T), S(i-1)))^2) & \text{if } S(i) > \max(\max(T), S(i-1)) \\ \min(v, (S(i) - \min(\min(T), S(i-1)))^2) & \text{if } S(i) < \min(\min(T), S(i-1)) \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Theorem 7. *For any two time series S and T of length L, a stiffness parameter $v \geq 0$ and a constant penalty $\lambda \geq 0$, and an alignment path $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_K\}$, where $\mathcal{A}_k = (i,j)$ indicates $S(i)$ is aligned to $T(j)$, the following inequality holds:* LB_TWED$(Q,C) \leq$ TWED(Q,C)

Proof. We can rewrite the equation for TWED using $\mathcal{A}$ as the following

$$\text{TWED}(S,T) = \sum_{k=1}^{K} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$$

where $\mathcal{A}_k^1 = i, \mathcal{A}_k^2 = j$, $\texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$ represents the cost of aligning $S(\mathcal{A}_k^1)$ to $T(\mathcal{A}_k^2)$ under TWED. From Equation 2.9, TWED is based on three different cost values.

$$\texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2)) =$$

$$\begin{cases} (S(i)-T(j))^2 + v|t^S(i) - t^T(j)| + (S(i-1) - T(j-1))^2 + v|t^S(i-1) - t^T(j-1)| \\ (S(i) - S(i-1))^2 + v|t^S(i) - t^S(i-1)| + \lambda \\ (T(j) - T(j-1))^2 + v|t^T(j) - t^T(j-1)| + \lambda \end{cases}$$

where $S(0) = 0$, $T(0) = 0$, $t^S(0) = 0$ and $t^T(0) = 0$. and we wish to proof the following

$$\text{LB\_TWED}(S,T) \leq \sum_{k=1}^{K} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$$

We know that $L \leq K$, so we can match every term on the LHS with a term on the RHS, giving $K - L$ terms unmatched.

$$\text{LB\_TWED}(S,T) \leq \sum_{k \in matched} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2)) + \sum_{k \in unmatched} \texttt{cost}(S(\mathcal{A}_k^1), T(\mathcal{A}_k^2))$$

Let us consider the relationship between matched terms on the RHS and the LHS terms. We wish to proof that for all $i > 0$, all the terms in Equation 5.4 is less than or equal to all the cost functions on the RHS. Due to boundary conditions, TWED must aligns the points at $i = 1$, $j = 1$. So for $i = 1$ it is trivial to see that,

$$\min \begin{cases} (S(1) - T(1))^2 \\ S(1)^2 + v + \lambda \\ T(1)^2 + v + \lambda \end{cases} \leq \begin{array}{c} (S(1) - T(1))^2 \\ (S(1) - 0)^2 + v \cdot t^S(1) + \lambda \\ (T(1) - 0)^2 + v \cdot t^T(1) + \lambda \end{array}$$

There are three cases for $i \geq 2$, and we shall start with the first one $S(i) > \max(\max(T), S(i-1))$, assuming $\max(T) \geq S(i-1)$ then $S(i) > \max(T)$. We will first show the proof by assuming $(S(i) - \max(T))^2 \leq v$.

- $(S(i) - \max(T))^2 \leq (S(i) - T(j))^2 + v|t^S(i) - t^T(j)| + (S(i-1) - T(j-1))^2 + v|t^S(i-1) - t^T(j-1)|$ because $\max(T) \geq T(j)$ implies that $T(j)$ is further from $S(i)$ than $\max(T)$, so $(S(i) - \max(T))^2 \leq (S(i) - T(j))^2$. All the other terms cannot be negative.

- $(S(i) - \max(T))^2 \leq (S(i) - S(i-1))^2 + v|t^S(i) - t^S(i-1)| + \lambda$ because $\max(T) \geq S(i-1)$ implies that $S(i-1)$ is further from $S(i)$ than $\max(T)$, so $(S(i) - \max(T))^2 \leq (S(i) - S(i-1))^2$. All the other terms cannot be negative.

- $(S(i) - \max(T))^2 \leq (T(j) - T(j-1))^2 + v|t^T(j) - t^T(j-1)| + \lambda$ because $(S(i) - \max(T))^2 \leq v$, $|t^T(j) - t^T(j-1)| > 0$ and all the other terms cannot be negative.

Assuming $v \leq (S(i) - \max(T))^2$, $v \leq (S(i) - T(j))^2$ because $(S(i) - \max(T))^2 \leq (S(i) - T(j))^2$. $v \leq |t^S(i) - t^S(i-1)|$ and $v \leq |t^T(j) - t^T(j-1)|$ because $|t(i) - t(i-1)| > 0$. Thus the proof still holds.

For the case $S(i-1) \geq \max(T)$ then $S(i) > S(i-1)$, and assuming $(S(i) - S(i-1))^2 \leq v$, the opposite $v \leq (S(i) - S(i-1))^2$ will still hold by applying the same reasoning.

- $(S(i) - S(i-1))^2 \leq (S(i) - T(j))^2 + v|t^S(i) - t^T(j)| + (S(i-1) - T(j-1))^2 + v|t^S(i-1) - t^T(j-1)|$ because $S(i) > S(i-1)$, $S(i-1) \geq \max(T)$ and $\max(T) \geq T(j)$ implies that $T(j)$ is further from $S(i)$ than $S(i-1)$ so $(S(i) - S(i-1))^2 \leq (S(i) - T(j))^2$. All the other terms cannot be negative.
- $(S(i) - S(i-1))^2 \leq (S(i) - S(i-1))^2 + v|t^S(i) - t^S(i-1)| + \lambda$ is trivial because all the terms cannot be negative.
- $(S(i)-S(i-1))^2 \leq (T(j)-T(j-1))^2 + v|t^T(j)-t^T(j-1)| + \lambda$ because $(S(i)-S(i-1))^2 \leq v$, $|t^T(j) - t^T(j-1)| > 0$ and all the other terms cannot be negative.

Similar proof can be applied to the second case $S(i) < \min(\min(T), S(i-1))$. The third case is trivial as all of the costs are non-negative. Since all the matched terms are larger than the LHS, the unmatched terms has to be negative for the inequality to be false. Fortunately, it is not possible because $v \geq 0$, $\lambda \geq 0$, $|t(i) - t(j)| \geq 0$ and the squared terms can never be negative. Therefore LB_TWED$(S, T) \leq$ TWED(S, T) holds. $\square$

5.4 FASTEE: FAST Ensembles of Elastic Distances

Given an elastic distance measure E, N training instances and M parameters to learn, learning the best parameter for E can be seen as creating a $N \times M$ table, similar to Table 5.2, giving the NN for every time series for all parameters. As demonstrated in [3], training just a single classifier (NN-DTW) is very computationally expensive. This is even more problematic when there are 11 classifiers to train. In this section, we introduce *Fast Ensembles* *of Elastic Distances* (FASTEE), an extension of FASTWWS to speed up the training time for EE.

5.4.1 Properties for Fast Elastic Ensemble

We can generalise the main properties of FASTWWS to other elastic distance measures. As indicated in Section 5.2.2, an elastic distance measure, E is typically parametrised by an alignment penalty, path constraint and threshold parameter. We let $\mathcal{P}^p$, $\mathcal{P}^c$ and $\mathcal{P}^t$ be the alignment penalty, path constraint and threshold parameter respectively.

Property #1: For any elastic distance measure E, it is monotone with its alignment penalty constraint parameters $\mathcal{P}^p$

Theorem 8. *Let E be the elastic distance measure of interest, S, T be two time series, $\mathcal{P}^p$ be an alignment penalty, $\hat{\mathcal{P}^p}$ the smaller penalty and $\mathcal{A}^{\mathcal{P}^p}$ the associated alignment path. Then we have $E_{\mathcal{P}^p}(S,T) \geqslant E_{\hat{\mathcal{P}^p}}(S,T)$.*

Proof. Let $\texttt{cost}_{\mathcal{P}^p}(S(i),T(j))$ be the cost of aligning the two points $S(i)$ and $T(j)$. If $\mathcal{P}^p \geq \hat{\mathcal{P}^p}$, then $\texttt{cost}_{\mathcal{P}^p}(S(i),T(j)) \geq \texttt{cost}_{\hat{\mathcal{P}^p}}(S(i),T(j))$. Otherwise, the alignment path will not be optimum. □

Property #2: For any elastic distance measure E, alignment path can be valid for several path constraint parameters $\mathcal{P}^c$

Theorem 9. *Let E be the elastic distance measure of interest, S, T be two time series, $\mathcal{P}^c_1$ and $\mathcal{P}^c_2$ two path constraint parameters and $\mathcal{A}^{\mathcal{P}^c_1}$ and $\mathcal{A}^{\mathcal{P}^c_2}$ their associated alignment paths. $\mathcal{A}^{\mathcal{P}^c_1} = \mathcal{A}^{\mathcal{P}^c_2} \Rightarrow E_{\mathcal{P}^c_1}(S,T) = E_{\mathcal{P}^c_2}(S,T)$. In other words, $E_{\mathcal{P}^c_1}(S,T)$ can only differ from $E_{\mathcal{P}^c_2}(S,T)$ if the alignment path differs.*

Proof. Let $\mathcal{A}^{\mathcal{P}_1^c} = \langle (i_1^{\mathcal{P}_1^c}, j_1^{\mathcal{P}_1^c}), \cdots, (i_K^{\mathcal{P}_1^c}, j_K^{\mathcal{P}_1^c}) \rangle$, $\mathcal{A}^{\mathcal{P}_2^c} = \langle (i_1^{\mathcal{P}_2^c}, j_1^{\mathcal{P}_2^c}), \cdots, (i_K^{\mathcal{P}_2^c}, j_K^{\mathcal{P}_2^c}) \rangle$. We have:

$$\begin{aligned} E_{\mathcal{P}_1^c}(S,T) &= \textstyle\sum_{k=1}^{K} \texttt{cost}(S(i_k^{\mathcal{P}_1^c}), T(j_k^{\mathcal{P}_1^c})) \qquad \text{Eq 5.1} \\ &= \textstyle\sum_{k=1}^{K} \texttt{cost}(S(i_k^{\mathcal{P}_2^c}), T(j_k^{\mathcal{P}_2^c})) \quad \text{(By hyp.)} \\ &= E_{\mathcal{P}_2^c}(S,T) \end{aligned}$$

□

Theorem 10. *Let E be the elastic distance measure of interest, S, T be two time series, $\mathcal{P}^c$ be a path constraint, $\hat{\mathcal{P}}^c$ the smaller path constraint and $\mathcal{A}^{\mathcal{P}^c}$ the associated alignment path. Then*

$$(|i_k - j_k| < \mathcal{P}^c)_{\forall k} \Rightarrow E_{\mathcal{P}^c}(S,T) = E_{\hat{\mathcal{P}}^c}(S,T)$$

In other words, if all the points in an alignment path are within the boundaries of the path constraint $\mathcal{P}^c$, then $E_{\mathcal{P}^c}(S,T) = E_{\hat{\mathcal{P}}^c}(S,T)$.

Proof. All elastic distance measures find an alignment path $\mathcal{A}^{\mathcal{P}^c}$ such that

$$\sum_{k=1}^{K} \texttt{cost}(S(i_k)T(j_k))$$

is minimized respecting the constraint $|i_k - j_k| \leqslant \mathcal{P}^c$. □

In FastWWS, this property is known as the "window validity" [3]. Thus, similarly we say that E$_{\mathcal{P}^c}(S,T)$ has a "path validity" if all the alignment paths are the same. Note that this property is valid only for a fixed alignment penalty parameter $\mathcal{P}^p$ and threshold parameter $\mathcal{P}^t$.

Property #3: For any elastic distance measure E, it is monotone with its path constraint parameters $\mathcal{P}^c$

Theorem 11. *Let E be the elastic distance measure of interest other than* TWED*, S, T be two time series, and $\mathcal{P}^c$ a parameter of E, $\hat{\mathcal{P}}^c < \mathcal{P}^c$, we have E$_{\mathcal{P}^c}(S,T) \leqslant$E$_{\hat{\mathcal{P}}^c}(S,T)$ and* TWED$_{\mathcal{P}^c}(S,T) \geqslant$ TWED$_{\hat{\mathcal{P}}^c}(S,T)$.

Proof. Assume $\mathsf{E}_{\mathcal{P}^c}(S,T) > \mathsf{E}_{\hat{\mathcal{P}^c}}(S,T)$, then this means that there exists an alignment path $\mathcal{A}_{\hat{\mathcal{P}^c}}$ such that the associated cost is lower than the one for $\mathcal{A}_{\mathcal{P}^c}$. This translates in E not having found the optimal solution at $\mathcal{P}^c$, which is a contradiction. $\square$

Note that larger path constraint parameter for TWED, v gives larger distance which is the opposite of the path constraint parameters for all the other elastic distance measures. Despite that, the proof is still applicable by just inverting the signs.

Property #4: For any elastic distance measure E with a path constraint $\mathcal{P}^c$, its lower bound (in this work) is monotone with $\mathcal{P}^c$

Theorem 12. *Let E be the elastic distance measure of interest except* TWED*, S, T be two time series, $\mathcal{P}^c$ a parameter of E, $\hat{\mathcal{P}^c} < \mathcal{P}^c$, and LB a lower bound of E used in this work, we have $LB_{\mathcal{P}^c}(S,T) \leqslant LB_{\hat{\mathcal{P}^c}}(S,T)$ and* $\text{LB\_TWED}_{\mathcal{P}^c}(S,T) \geqslant \text{LB\_TWED}_{\hat{\mathcal{P}^c}}$*.*

Proof. To proof this property, we have to look at each of the lower bounds for distances with path constraint parameter – DTW, ERP, LCSS and TWED. The proof for DTW can be found in [3]. The lower bound for ERP is an adaptation of LB_KEOGH for DTW. Thus, its proof is the same as DTW.

The lower bound for LCSS is bounded by an upper $\mathbb{UE}$ and lower $\mathbb{LE}$ envelopes. Let $\mathcal{P}^c = \Delta$, $\mathbb{UE}_\Delta(i) = \max(q_{i-\Delta} : q_{i+\Delta}) + \varepsilon$ be the elements in the upper envelope and $\mathbb{LE}_\Delta(i) = \min(S(i-\Delta) : S(i+\Delta) - \varepsilon$ as the lower envelope. We wish to proof

$$\text{LB\_LCSS}_\Delta(S,T) \leqslant \text{LB\_LCSS}_{\Delta-1}(S,T)$$

We will assume the opposite and show that it leads to a contradiction:

$$\text{LB\_LCSS}_{\Delta}(S,T) > \text{LB\_LCSS}_{\Delta-1}(S,T)$$

$$1-\frac{1}{L}\sum_{i=1}^{L}\begin{cases}1 & \text{if } \mathbb{LE}_{\Delta}(i) \leq S(i) \leq \mathbb{UE}_{\Delta}(i)\\ 0 & \text{otherwise}\end{cases} > 1-\frac{1}{L}\sum_{i=1}^{L}\begin{cases}1 & \text{if } \mathbb{LE}_{\Delta-1}(i) \leq S(i) \leq \mathbb{UE}_{\Delta-1}(i)\\ 0 & \text{otherwise}\end{cases}$$

$$\sum_{i=1}^{L}\begin{cases}1 & \text{if } \mathbb{LE}_{\Delta}(i) \leq S(i) \leq \mathbb{UE}_{\Delta}(i)\\ 0 & \text{otherwise}\end{cases} < \sum_{i=1}^{L}\begin{cases}1 & \text{if } \mathbb{LE}_{\Delta-1}(i) \leq S(i) \leq \mathbb{UE}_{\Delta-1}(i)\\ 0 & \text{otherwise}\end{cases}$$

The above implies that larger envelope at $\mathcal{P}^c$ gives a larger lower bound distance than smaller envelopes, which contradicts with Equation 2.24 that defines LB_LCSS as the percentage of points that are inside the envelope. When $\mathcal{P}^c$ increases, the envelope gets larger, thus more points from S will fall into the envelope. Hence, the number of $S(i)$ points in the larger envelopes will be larger than the smaller envelopes giving longer common subsequence and consequently a smaller distance.

TWED monotonically increases with its path constraint parameter $\mathcal{P}^c = v$ and we let $v' < v$. Since $v > v'$ and the rest of the Equation 5.4 is constant regardless of v, it is trivial to see that the inequalities for all the terms in Equation 5.4 holds.

$$\text{LB\_TWED}_{v}(S,T) \geqslant \text{LB\_TWED}_{v'}(S,T)$$

$$\min\begin{cases}(S(1)-T(1))^2\\ S(1)^2+v+\lambda\\ T(1)^2+v+\lambda\end{cases} \geqslant \min\begin{cases}(S(1)-T(1))^2\\ S(1)^2+v'+\lambda\\ T(1)^2+v'+\lambda\end{cases}$$

$$\sum_{i=2}^{L}\begin{cases}\min(v,(S(i)-A)^2) & \text{if } S(i) > A\\ \min(v,(S(i)-B)^2) & \text{if } S(i) < B\\ 0 & \text{otherwise}\end{cases} \geqslant \sum_{i=2}^{L}\begin{cases}\min(v',(S(i)-A)^2) & \text{if } S(i) > A\\ \min(v',(S(i)-B)^2) & \text{if } S(i) < B\\ 0 & \text{otherwise}\end{cases}$$

where $A = \max(\max(T), S(i-1))$, $B = \min(\min(T), S(i-1))$. □

Property #5: For any elastic distance measure E with a penalty parameter $\mathcal{P}^p$, its lower bound (in this work) is monotone with $\mathcal{P}^p$

Theorem 13. *Let E be the elastic distance measure of interest, S, T be two time series, $\mathcal{P}^p$ a parameter of E, $\hat{\mathcal{P}}^p < \mathcal{P}^p$, and LB a lower bound of E used in this work, we have $LB_{\mathcal{P}^p}(S,T) \geqslant LB_{\hat{\mathcal{P}}^p}(S,T)$.*

Proof. Since the penalty parameter $\mathcal{P}^p \geq 0$ and all lower bounds for elastic distance measures are additive, it is trivial that $LB_{\mathcal{P}^p}(S,T) \geqslant LB_{\hat{\mathcal{P}}^p}(S,T)$ because $\mathcal{P}^p > \hat{\mathcal{P}}^p$ □

Note that LB_WDTW is not considered although WDTW has a penalty parameter g which controls the level or penalisation for far points using Equation 2.4. This is because LB_WDTW only considers the minimum weight, $\mathtt{w}_0$ which is invariant to g.

Property #6: For any elastic distance measure E with a threshold parameter $\mathcal{P}^t$, its lower bound (in this work) is monotone with $\mathcal{P}^t$

Theorem 14. *Let E be the elastic distance measure of interest, S, T be two time series, $\mathcal{P}^t$ a parameter of E, $\hat{\mathcal{P}}^t < \mathcal{P}^t$, and LB a lower bound of E used in this work, we have $LB_{\mathcal{P}^t}(S,T) \geqslant LB_{\hat{\mathcal{P}}^t}(S,T)$.*

Proof. Considering the elastic distance measures used in this work, only the LCSS distance has the threshold parameter, thus the proof will be based on LB_LCSS. Let $\mathcal{P}^t = \varepsilon$, LB_LCSS is bounded by an upper $\mathbb{UE}(i) = \max(T(i-\Delta) : T(i+\Delta)) + \varepsilon$ and lower $\mathbb{LE}(i) = \min(T(i-\Delta) : T(i+\Delta)) - \varepsilon$ envelopes which is built based on ε. From Equation 2.24, LB_LCSS is defined as the percentage of points that are inside the envelope. When $\mathcal{P}^t$ increases, the envelope gets larger, thus more points from S will fall into the envelope. Hence, the number of points in the larger envelopes will be larger than the smaller envelopes giving longer common subsequence and consequently a smaller distance. □

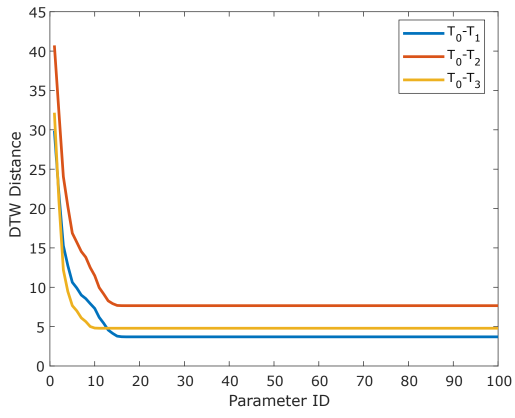
Figure 5.3 and 5.4 illustrate the combination of all the properties for each of the elastic distance measures. In this work, to be consistent with EE, only 100 parameters are chosen for each of the distances [23]. Figure 5.3 shows the properties for elastic distance measures with a single parameter, while Figure 5.4 with two parameters. For distances with two parameters, 10 values are chosen uniformly for each of the parameters, creating a combination of 100 parameter values. There are no distances used with three parameters. Note that the nearest neighbour (lowest distance) changes as the parameter changes.

DTW, DDTW, WDTW and WDDTW monotonically decreases with increasing parameter as shown in Figure 5.3a, 5.3b, 5.3c and 5.3d. Figure 5.3c and 5.3d only show 15 parameters as the WDTW distances are very small for larger parameters. The parameter for DTW and DDTW – warping window w is selected from the range $[0, L]$ with an increment of $0.01 \times L$. The parameter g for WDTW and WDDTW is chosen from an uniform distribution of $U(0, 1)$ with 100 values. MSM monotonically increases with its parameter as shown in Figure 5.3e. Its parameter `c` is sampled from an exponential sequence in the range $[0.01, 100]$ [23].

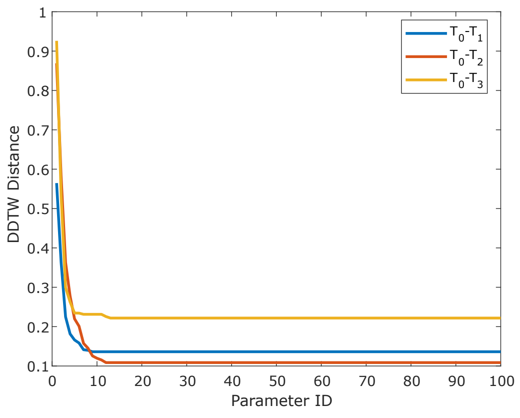
LCSS has a monotonically decreasing relationship with its parameters as shown in Figure 5.4a. The threshold parameter for LCSS, ε is chosen from the range $[\sigma/5, \sigma]$ where σ is the standard deviation of the training set. The path constraint parameter, δ is chosen from the range of $[0, L/4]$ [23]. Figure 5.4b shows the monotonically increasing relationship of TWED and its parameters. The constraint parameter for TWED, v is chosen from an exponential distribution ranging from $[10^{-5}, 1]$. The penalty parameter λ is chosen uniformly from the range $[0, 0.1]$. ERP distance increases when its alignment penalty g increases and decreases when its path constraint, `bandsize` increases as shown in Figure 5.4c. Both of its parameters are chosen the same way as LCSS.

5.4.2 The FastEE Algorithms

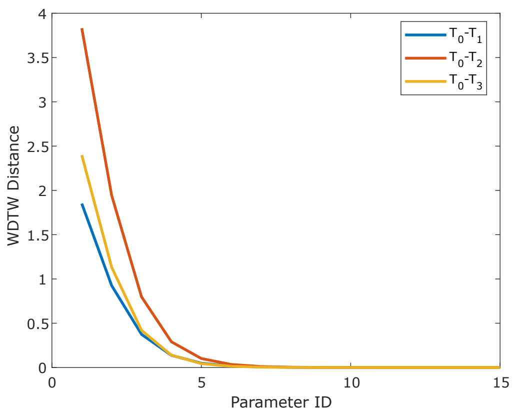
The previous sections show the theoretical basis of our work. We are now in a better position to explain our algorithms. The FastEE algorithm applies the strategy that underlies



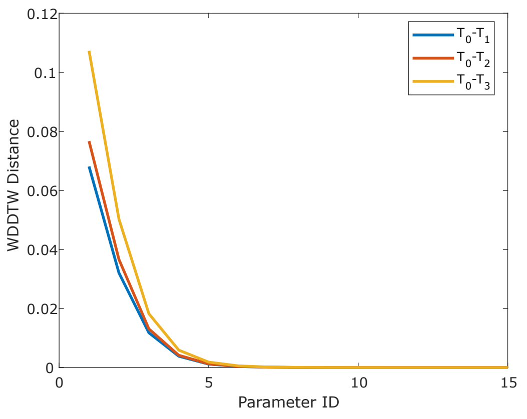
(a) DTW



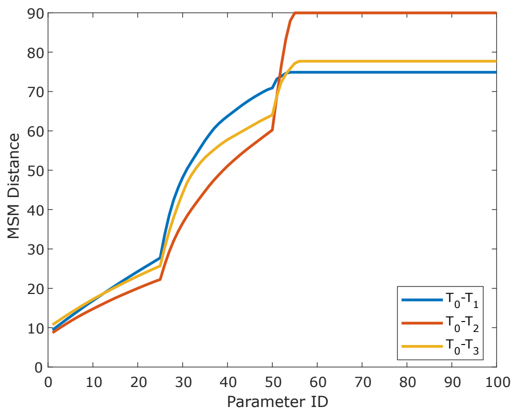
(b) DDTW



(c) WDTW



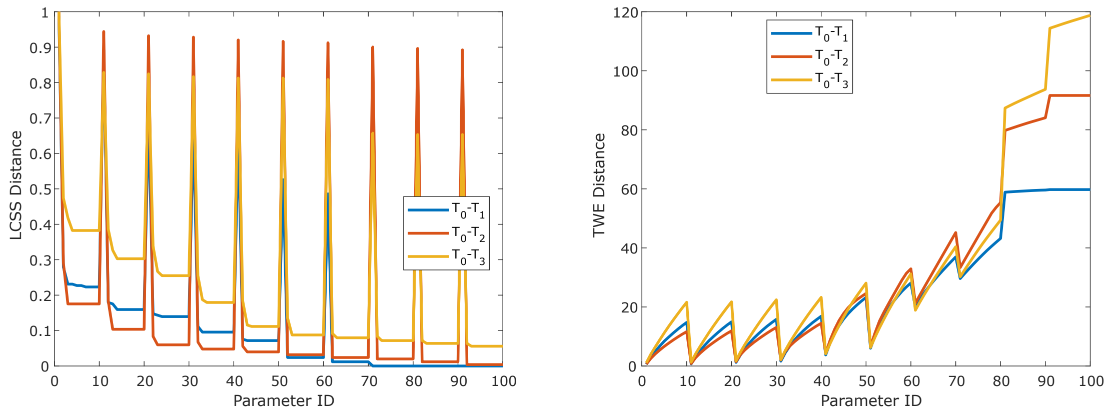
(d) WDDTW



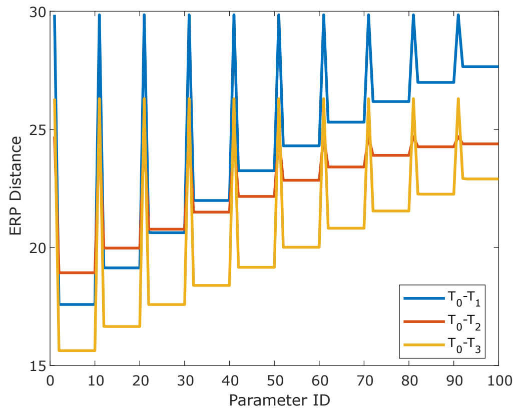
(e) MSM

Figure 5.3: Relationship between single parameter elastic distances and their parameters

FastWWS [3] to all the components of EE. Like FastWWS, FastEE is an exact algorithm that is capable of giving the exact best parameter (with respect to LOO evaluation on the training data) and can also be applied to a subset of parameters. To take advantage of all the properties of FastEE, we order the computations across the columns of the NNs table systematically for each distance measure. This allows FastEE to prune most of the computations across the columns of the table.



(a) LCSS (b) TWED



(c) ERP

Figure 5.4: Relationship between double parameter elastic distances and their parameters

FastEE starts scanning the parameter values for DTW, DDTW, WDTW and WDDTW from the largest to the smallest. Distances at larger parameter values can be used as a lower bound to the smaller parameter. Both DTW and DDTW distances are constant at larger windows as illustrated in Figure 5.3a and 5.3b. This has the effect of pruning the computations for DTW and DDTW at the windows where they are constant. The sigmoid weight function makes WDTW and WDDTW a continuous function as illustrated in Figure 5.3c and 5.3d, preventing them from having a constant value. Since it does not satisfy property 2, the computations of WDTW cannot be pruned. For this reason, FastEE will only make use of the distances computed at a larger g as the lower bounds for smaller g. Lower bounds for WDTW are also used to speed up the process.

As MSM has a monotonically increasing relationship with its parameter c, FastEE starts from the smallest c – using distances at smaller c as a lower bound for larger c. Note that at larger c, MSM is constant and thus the computations can be pruned.

The *spikes* in Figure 5.4 correspond to the changeover of a new penalty and threshold parameter for elastic distances with two parameters. Currently, FASTEE resets the scan at the *spikes*. We note that it is possible to use the distances at larger penalty value as the lower bound to a smaller penalty value but this exploration will be left for future work.

FASTEE starts from the largest parameter combination (Δ, ε) for LCSS. For a fixed ε, LCSS distance stays constant for a range of Δ. Thus the computation can be pruned when the distance is constant. Smaller threshold ε means that less points will be a match and thus larger distance. Hence, LCSS distance at larger ε can be used as the lower bound at smaller ε.

For TWED, FASTEE starts scanning from the smallest parameter combination (v, λ). It is interesting to see that in Figure 5.4b, TWED is continuous at smaller v and does not satisfy property 2. However, it remains constant at larger v. Hence FASTEE is still applicable to TWED but will not yield good speed up at these smaller v.

Figure 5.4c shows that the ERP distance decreases with increasing `bandsize`, FASTEE starts searching from the largest parameter combination and resets at every new g. Larger g corresponds to larger penalty and thus larger ERP distance. It is important to note that ERP at $\texttt{bandsize} = 0$ has the same value regardless of g which means that we can reuse this value.

Lazy Nearest Neighbour Assessment

Our LAZYASSESSNN algorithm, presented in Algorithm 14, generalizes FASTWWS so that it can be used for all EE's elastic distance measures. It assesses whether a pair of time series can be less than a distance d apart for a given parameter $\mathcal{P}$. LAZYASSESSNN assesses each of the potential nearest neighbours in a *lazy* fashion, by making the most out of all possible lower bounds. It is lazy in that it postpones calculations for as long as possible. The ordering of the elastic distance measure computations from small to large allows any value previously calculated to become a lower bound to the current parameter value. A cache $\mathcal{C}$ is used to store the results from the previous parameter value.

Algorithm 14: LAZYASSESSNN($\mathcal{C}_{(S,T)}, \mathcal{P}, d, S, T$)

```
Input: $\mathcal{C}_{(S,T)}$: cache storing the previous measure between $S$ and $T$
Input: $\mathcal{P}$: parameter
Input: $d$: the distance to beat
Input: $S, T$: the time series to measure
Result: $E_{\mathcal{P}}(S,T)$ if $\geqslant d$, else pruned
1  if $\mathcal{C}_{(S,T)} = \varnothing$ then $\mathcal{C}_{(S,T)} \leftarrow$ init($S,T$)
2  switch $\mathcal{C}_{(S,T)}$.stoppedAt do
       // $LB$ calculated with previous parameter $\hat{\mathcal{P}}$
3      case $E_{\hat{\mathcal{P}}}$ do
4          if $\mathcal{P} \in \mathcal{C}_{(S,T)}$.valid $\wedge$ $\mathcal{C}_{(S,T)}$.value $< d$ then
5              return $\mathcal{C}_{(S,T)}$.value
6          end
7      case $LB_{\hat{\mathcal{P}}}$ do
8          if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
9      otherwise do
           // Calculate $LB$ and $E$ at $\mathcal{P}$
           // Possible to cascade $LB$s for more than 1 lower bound starting
              from the lowest complexity
10         $\mathcal{C}_{(S,T)} \leftarrow LB_{\mathcal{P}}(S,T)$
11         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
12         $\mathcal{C}_{(S,T)} \leftarrow E_{\mathcal{P}}(S,T)$
13         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
14         return $\mathcal{C}_{(S,T)}$.value
15     end
16 end
```

First, we have to initialise the cache if it was not initialised previously. For most elastic distance measures, the initialisation is simply creating the cache. But for DTW, DDTW and ERP the cache is initialised with the LB_KIM distance. The reason is that LB_KIM is not well-defined for the other distances. LB_KIM is the loosest and cheapest lower bound to compute. It is sufficient to filter out the obvious unpromising candidates in the training set $\mathcal{T}$. Then we test where the cache last stopped, i.e. was it computing a lower bound for the target parameter value $\mathcal{P}$, was it computing a lower bound for previous parameter value $\hat{\mathcal{P}}$, or was it computing a distance for previous parameter value $E_{\hat{\mathcal{P}}}$. If it stopped at $E_{\hat{\mathcal{P}}}$, then we have to assess if $E_{\hat{\mathcal{P}}}$ is still valid and having a value less than d. On line 7-9, if we cannot prune with $E_{\hat{\mathcal{P}}}$, then we check if we can prune using previously computed bounds. Otherwise, we have to compute the lower bound for the target parameter and test if we can prune them. Note that we can cascade the bounds for distances with more than one bound. The bounds are ordered by their complexity which usually corresponds to their

```
Algorithm 15: FastEE($\mathcal{T}, C$)
Data: $\mathcal{T}$: training data
Data: $C$: set of NN classifier paired with an elastic distance
Result: $\mathcal{P}^{\star}$: best parameter for each distances
Result: $bestAccuracy$: best LOO-CV accuracy for each distances
foreach $c_i \in C$ do
    $\mathcal{P}^i \leftarrow \{\mathcal{P}^i_1, ..., \mathcal{P}^i_M\}$
    NNs $\leftarrow c_i$.FastFillNNTable($\mathcal{T}, \mathcal{P}^i$)
    $bestAccuracy \leftarrow -1$
    for $\mathcal{P}^i_p \leftarrow \mathcal{P}^i_1$ to $\mathcal{P}^i_M$ do
        $nCorrect \leftarrow 0$
        foreach $T_t \in \mathcal{T}$ do
            if NNs$[t][p]$.class $\neq T_t$.class then
                $nCorrect$++
            end
        end
        if $nCorrect > bestAccuracy$ then
            $bestAccuracy \leftarrow nCorrect$
            $\mathcal{P}^{\star}_i \leftarrow \mathcal{P}^i_p$
        end
    end
end
```

tightness [3, 32]. In this work, we cascade DTW lower bounds in the order – LB_KIM(S, T), LB_KEOGH(S, T) and LB_KEOGH(T, S). Reversing the role of S and T in LB_KEOGH can sometimes provide better bounds [32]. Finally if all bounds failed to prune the candidate, then we have to compute $E_{\mathcal{P}}$ – the elastic distance measure E at the target parameter $\mathcal{P}$.

FASTEE Algorithm

Recall that the problem of learning the best parameter for an elastic distance measure, E can be re-framed into creating a $(N \times M)$ NNs table, which gives the NN of every time series in the training set for all M parameters. Such a table is depicted in Table 5.2 (in Section 5.2.3). Once this table is built, the best parameter can be learned in one pass over it as described in Algorithm 15. The algorithm uses Algorithm 16 to fill the table and returns the parameter value with the highest LOO-CV accuracy on the training set $\mathcal{T}$ of size N.

The core of FASTEE actually depends on how efficient we can compute this table. Algorithm 16 describes how we build this table efficiently for a particular elastic distance measure.

At the highest level, the algorithm builds this table for a subset $\mathcal{T}' \subseteq \mathcal{T}$ of increasing size until $\mathcal{T}' = \mathcal{T}$. For example we start by building the table for $\mathcal{T}$ comprising of only time series T_1 and T_2 and fill the table as if $\mathcal{T}$ is the entire dataset. It is trivial to see that T_1 is the nearest neighbour for T_2 and vice versa. Then a third time series T_3 is added to the set $\mathcal{T}'$ from $\mathcal{T} \setminus \mathcal{T}'$. Now we have to find the nearest neighbour for T_3 from $\mathcal{T}' \setminus T_3 = \{T_1, T_2\}$ and check if T_3 is the nearest neighbour for both T_1 and T_2. This process is repeated until $\mathcal{T}' = \mathcal{T}$.

Algorithm 16 starts by initialising the $N \times M$, NNs table to $(\_, +\infty)$. This means that the table is empty and the distances are $+\infty$. The iteration starts from 2 in line 4 as there need to be at least 2 time series. Lines 5 to 7 are some initialisations including creating the cache associated with S to store the results. After initialisation, we start the NN search with the set of parameters $\bar{\mathcal{P}}_p$ in the order mentioned in the previous section. Then in line 9, we check if S already has a NN found from previous parameters. If S already has a NN, then we only need to check if S is the NN for the other time series in $\mathcal{T}'$. At this point, the LazyAssessNN algorithm assess if S "beats" the previous NN for each of the time series in $\mathcal{T}'$. If LazyAssessNN exits with `pruned` then it means that S is not the NN, otherwise the NNs table has to be updated with S as the new NN.

If we do not have the NN for S from the previous parameter value, then we will need to analyse all $(S, T)_{T \in \mathcal{T}'}$ and update the NNs table simultaneously for S and T. At this stage, we already have some information stored in the cache $\mathcal{C}$ about which $T \in \mathcal{T}'$ might be a better NN candidate for S. Note that the number of computations will be minimised if the first T is actually the NN of S. Hence, it is important to first assess the candidate with the highest NN potential by ordering the candidates. This method has been previously studied in [1] and used in FastWWS [3]. As mentioned in [3], it is possible that $\mathcal{C}$ contains different type of lower bounds leading to distances with different magnitude. Thus the lower bounds have to be normalised. Most of the lower bounds for the elastic distance measures are of $O(L)$ complexity, so we normalise them by the number of point-wise calculations. Elastic distance measures are then being normalised by a factor of $0.8/L$ which pushes the distances forward. This is because $E_{\mathcal{P}'}$ represents a better estimate of $E_{\mathcal{P}}$ than its lower bound.

```
Algorithm 16: FastFillNNTable($\mathcal{T}, \bar{\mathcal{P}}$)
Input: $\mathcal{T}$ the set of time series
Input: $\bar{\mathcal{P}}$ ordered parameters to scan
Result: $\mathrm{NNs}$ the nearest neighbors table
1  Define LANN as LazyAssessNN
2  $\mathrm{NNs}$.fillAll $(\_, +\infty)$
3  $\mathcal{T}' \leftarrow \emptyset$
4  for $s \leftarrow 2$ to $N$ do
5      $S \leftarrow \mathcal{T}_s$
6      $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\mathcal{T}_{s-1}\}$
7      foreach $T \in \mathcal{T}'$ do $\mathcal{C}_{S,T} \leftarrow \varnothing$
8      for $p \leftarrow M$ down to $1$ do
9          if $\mathrm{NNs}[s][p] \neq \varnothing$ then
               // Update table $\mathrm{NNs}[t][p]_{1 \leqslant t \leqslant s-1}$
10             for $t \leftarrow 1$ to $s-1$ do
11                 $toBeat \leftarrow \mathrm{NNs}[t][p]$.distance
12                 $res \leftarrow$ LANN$(\mathcal{C}_{(S,T_t)}, \bar{\mathcal{P}}_p, toBeat, S, T_t)$
13                 if $res \neq$ pruned then
14                     $\mathrm{NNs}[c][p] \leftarrow (S, res)$
15                 end
16             end
17         else
               // Check $S$ against previous $T \in \mathcal{T}'$
18             foreach $T_t \in \mathcal{T}'$ in asc. order using $\mathcal{C}$ do
19                 $toBeat \leftarrow \mathrm{NNs}[s][p]$.distance
20                 $res \leftarrow$ LANN$(\mathcal{C}_{(S,T_t)}, \bar{\mathcal{P}}_p, toBeat, S, T_t)$
21                 if $res \neq$ pruned then
22                     $\mathrm{NNs}[s][p] \leftarrow (T_t, res)$
23                 end
24                 $toBeatT \leftarrow \mathrm{NNs}[t][p]$.distance
25                 $resT \leftarrow$ LANN$(\mathcal{C}_{(S,T_t)}, \bar{\mathcal{P}}_p, toBeatT, S, T_t)$
26                 if $resT \neq$ pruned then
27                     $\mathrm{NNs}[t][p] \leftarrow (S, resT)$
28                 end
29             end
               // Propagate NN for all valid parameters
30             for $p' \in \mathrm{NNs}[s][p]$.valid do $\mathrm{NNs}[s][p'] \leftarrow \mathrm{NNs}[s][p]$
31         end
32     end
33 end
```

Line 19 gives the distance threshold from NNs$[s][p]$ where each candidate has to beat in order to be the NN of S. The candidate is assessed using the LazyAssessNN algorithm in Algorithm 14. Initially, this value will be ∞ as $\mathrm{NNs}[s][p] = \emptyset$ and a distance computation has to be done which will then be stored into $\mathrm{NNs}[s][p]$ in line 22. Then we check if S is the NN of each candidate $T \in \mathcal{T}'$. Finally after all $T \in \mathcal{T}'$ have been processed, NNS$[s][p]$ contains the actual NN of S at the parameter value $\mathcal{P}_p$. This information is then propagated across all $\mathcal{P}$ where the distance is valid.

5.5 Experiments

This section describes the experiments that evaluate our FASTEE algorithm. Our experiments were performed using all the 85 freely available benchmark UCR time series datasets and the original train/test split [6]. We performed a search over the 100 parameters specified in Section 5.4.1. We conducted all of our experiments on a 16 core Xeon-E5-2667-v3 @3.20GHz machine with 16GB RAM. Our source code has been made open-source at https://github.com/ChangWeiTan/FastEE and the full results at http://bit.ly/FastEE.

5.5.1 Speed-up Against EE

We first perform an experiment comparing FASTEE to the following:

- **EE** [17]: The standard implementation of the EE classifier. This naïve version is used as the baseline. We use the code from [17].
- **LBEE**: EE with lower bounds. This is the improvised version of naïve EE. It uses lower bounds for the elastic distance measures in all the NN search. NN search with lower bound has a lot of success with speeding up NN-DTW [27, 32].

All methods are exact – they all learn the same best parameter, the same LOO-CV accuracy and thus the same classification accuracy. This is shown in Figure 5.9b which shows the classification accuracy of FASTEE compared to EE. Hence our experiments are more focused on the training time.

Figure 5.5 compares the training time of EE (x-axis) to LBEE and FASTEE. Points under the red line indicate that the method is faster than standard EE. The result is significant and it shows that FASTEE is faster than the standard implementation of EE with all the red points

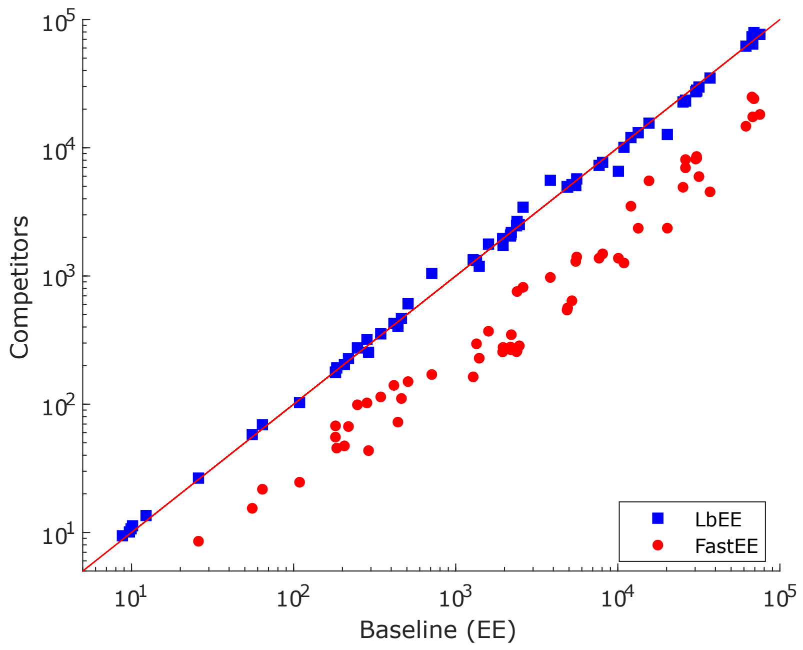
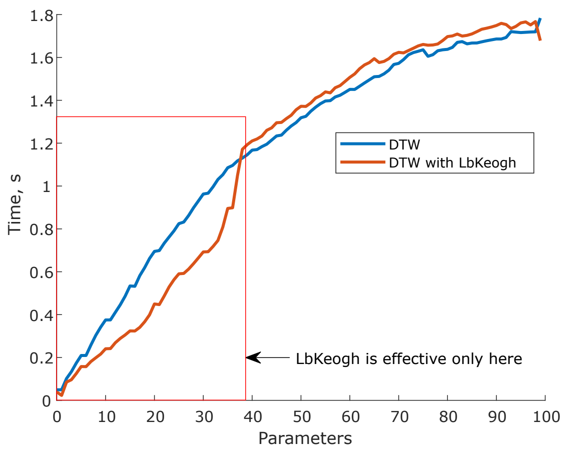


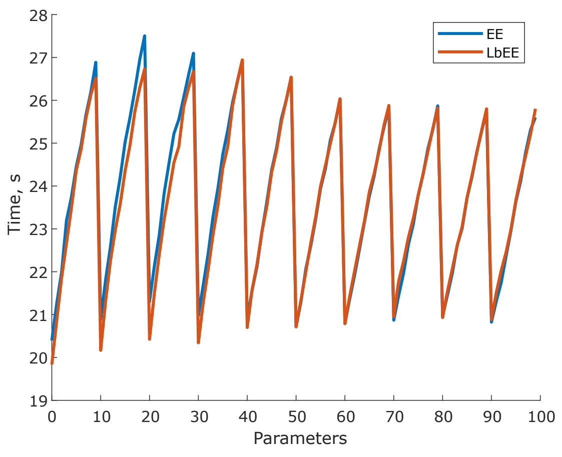
Figure 5.5: Total training time on the benchmark datasets (better seen in color)

consistently under the red line. The result in Figure 5.5 shows that using lower bounds on the elastic distance measures has no effect in reducing the training time, and sometimes increases it. This is a surprising result because lower bounds have proven to be successful in speeding up many time series tasks, such as NN-DTW similarity search [1, 27, 32]. We believe that there are two main reasons for this. For simplicity, we will explain the reasons using the DTW distance. The same reasoning applies to all other elastic distance measures as well.

First is due to the tightness of the lower bounds which is highly dependent on the parameters, especially the path constraint parameter. All lower bounds used in this work are similar to LB_KEOGH. They build an envelope to encapsulate the candidate time series (shown in Figure 2.6b), and sum the distances of all the points in the query time series that fall outside of the envelope. They are designed to ensure that if a point of a query time series that is outside of the envelope is compared to either the upper or lower envelope, the distance between them must at least contribute to the actual distance computation. For instance, the path constraint parameter affects the envelope in the horizontal direction, i.e. larger path constraint makes the envelope wider and bigger. Similarly larger penalty or threshold parameter increases the vertical direction, making the envelope taller and bigger. If the envelope



(a)



(b)

Figure 5.6: (a) Training time of DTW and DTW with LB_KEOGH (b) EE and LBEE on the `ProximalPhalanxOutlineAgeGroup` dataset [6]

is bigger, fewer points will be outside of the envelope. As a consequence, the lower bound has a smaller distance and thus decrease in tightness.

This is illustrated in Figure 5.6a, which compares the training time for NN-DTW and NN-DTW with LB_KEOGH at different parameters. Initially, LB_KEOGH is effective at the small warping windows but loses its effectiveness in pruning NN candidates as the warping window gets larger. In other words, computing lower bounds for DTW at these larger windows becomes redundant. A similar result is observed for the other elastic distance measures.

Second, using lower bounds across the columns of the NNs table is not efficient as they lose their tightness and need to be recomputed multiple times. As shown in Figure 4.3 and Figure 5.3a, DTW distance monotonically increases as the warping window decreases and is constant for a wide range of warping windows. This allows FASTEE to use DTW from larger windows as the lower bound for a smaller window to skip as many computations as possible and preventing from recomputing DTW at parameters that give the same value. LBEE does not make use of this information and thus has to recompute the distances multiple times. As a consequence of these two reasons, when all the distances are combined, on average the training time of EE does not improve as shown in Figure 5.6b.

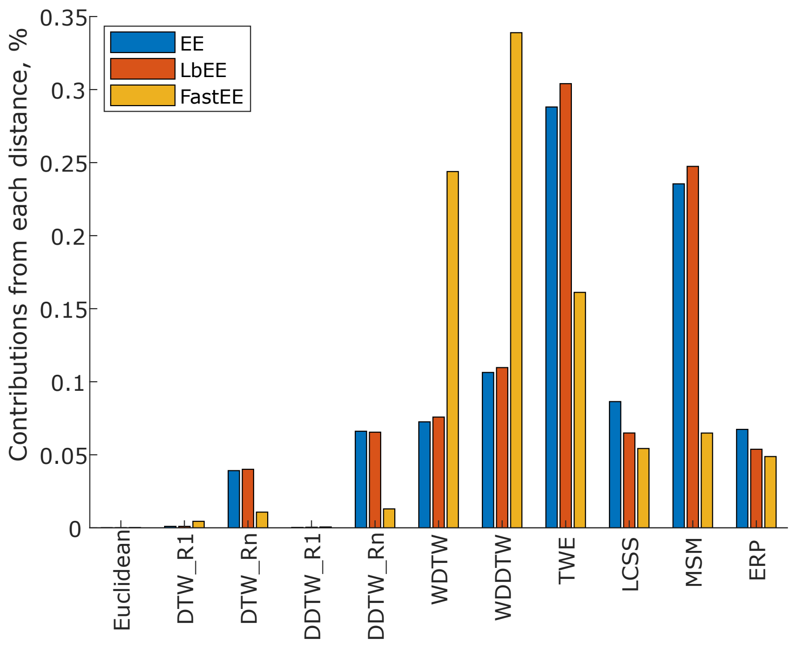


Figure 5.7: Contributions from each elastic distance measures to the total training time

5.5.2 Can FASTEE be Further Sped Up?

FASTEE is at most 10 times faster than the standard EE. This brings up the question of whether we can further reduce the training time of FASTEE.

Figure 5.7 shows the average contribution of each elastic distance measure to EE's total training time across the UCR benchmark archive [6]. TWED, MSM and WDDTW are the 3 distances that contribute the most to the training time of EE and LBEE. FASTEE significantly reduces the training time of MSM, which leaves WDDTW, WDTW and TWED the top 3 for FASTEE.

Recall that learning the best parameter for each elastic distance measures is typically done with LOO-CV and can be re-framed as filling up a $N \times M$ NNs table. This means that for each $T \in \mathcal{T}$, we search for its nearest neighbour from $\mathcal{T} \setminus T$. However, it is possible that we do no need the full N instances to learn the best parameter. In other words, we want to estimate the best parameter (which could be slightly different) based on a few instances $\mathcal{N}$ from the training set without compromising the classification accuracy. Hence, instead of $N \times M$, we wish to build a $\mathcal{N} \times M$ NNs table where $\mathcal{N} \ll N$. This has the consequence of speeding up the training time since less training instances are examined.

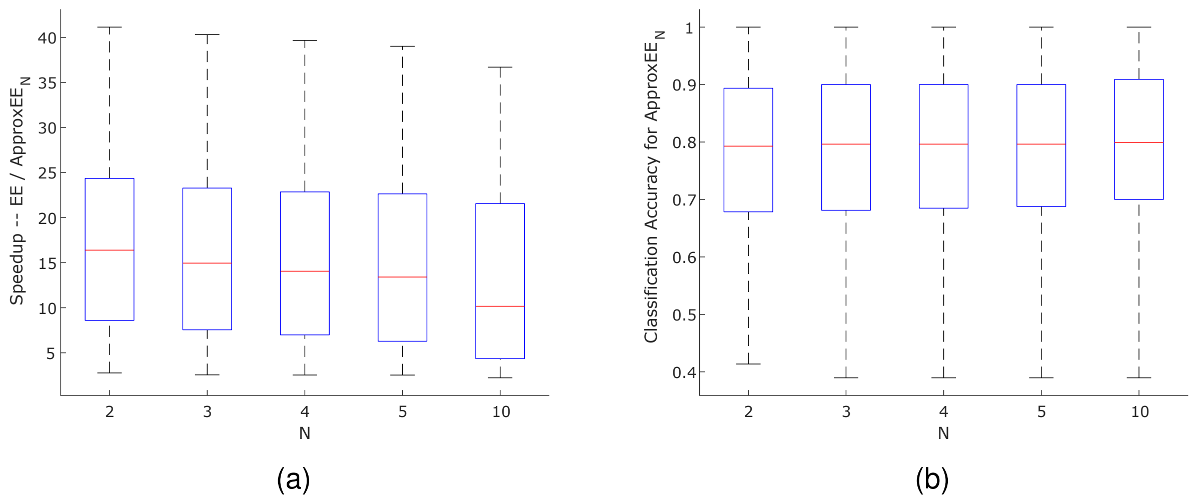


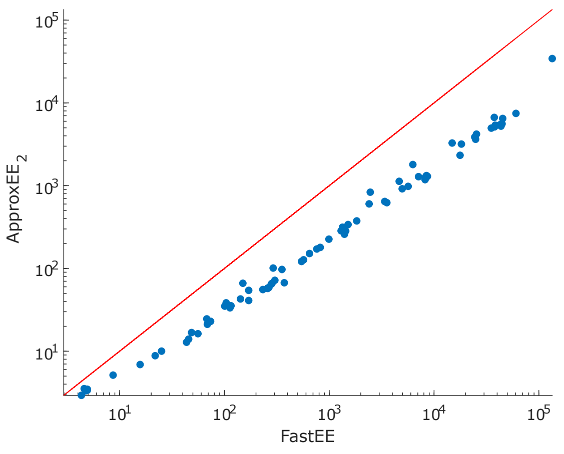
Figure 5.8: (a) Speedup against EE and (b) classification accuracy for APPROXEE across all the UCR benchmark datasets [6] for all the $\mathcal{N}$

A similar method has been proposed to speed up the training time of a classifier [103]. The authors proposed a contract algorithm to build the Random Interval Spectral Ensemble (RISE) classifier without compromising the classification accuracy. The authors estimate the interval r in RISE, based on the remaining contract time. They used a least squares linear regression model, which models the relationship of the interval r with training time. Note that it is possible to implement a contract version of FASTEE but this will be left for future work.

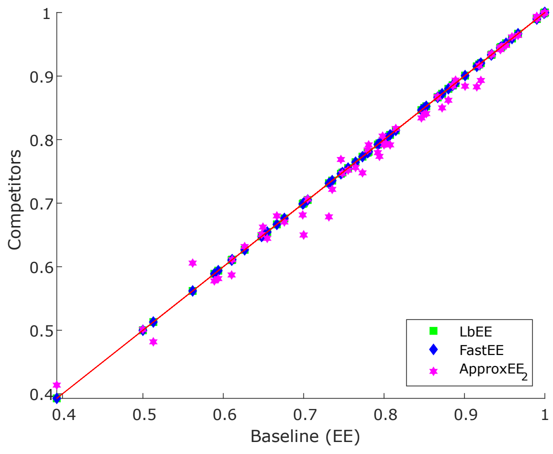
In this work, we propose a simple technique to approximate the LOO-CV process while not affecting the classification accuracy. Our technique – APPROXEE builds a $\mathcal{N} \times M$ table but using the full $\mathcal{T}$ of size N, i.e. we still search for the nearest neighbour from the full training set $\mathcal{T}$. This ensures that the nearest neighbour for each of the $\mathcal{N}$ is the same as exact LOO-CV – giving a better estimate of the parameters. Currently only the three distances – TWED, WDTW and WDDTW are approximated as they contribute the most to the training time of FASTEE. Approximating other distances is possible but the effect will not be great because they do not contribute much to the total training time.

We report the average training time and classification accuracy of APPROXEE at $\mathcal{N} = \{2, 3, 4, 5, 10\}$ over 5 runs and we write APPROXEE$_{\mathcal{N}}$. Note that $\mathcal{N} = 1$ was not tested because it does not make sense to train with just a single instance.

Figure 5.8a compares the speed up of APPROXEE against EE across all the UCR bench-



(a)



(b)

Figure 5.9: Average (a) training time of FASTEE and APPROXEE$_2$ and (b) classification accuracy of the different EE classifiers over 5 runs

mark datasets [6] for all $\mathcal{N}$. The result is expected as more time is required for training as $\mathcal{N}$ tends to N. This is indicated by the decreasing median (red line in the middle of the box plot) over multiple $\mathcal{N}$. Note that the training time across all the $\mathcal{N}$ is not significantly different. Since APPROXEE estimates the exact best parameter and the training time is similar for all $\mathcal{N}$, we are interested to know the effect on classification accuracy. Figure 5.8b compares the classification accuracy of APPROXEE across all the UCR benchmark datasets [6] for all $\mathcal{N}$. The result shows that there is no significant difference for all $\mathcal{N}$. This suggests that a small $\mathcal{N}$ is sufficient to provide a good speed up to both FASTEE and EE without compromising the classification accuracy.

Therefore, we choose $\mathcal{N} = 2$ and compare APPROXEE$_2$ to FASTEE. Figure 5.9a compares the total training time of APPROXEE to FASTEE. It shows that APPROXEE is always much faster than FASTEE. This is expected because $\mathcal{N} \ll N$. The largest speed-up gained from APPROXEE is 40 times on the `ElectricDevices` dataset where FASTEE is only 10 times faster than EE.

Finally we show the classification accuracy of the different EE classifiers in Figure 5.9b. As expected, the classification accuracy of LBEE and FASTEE is exactly the same as EE because they are exact, i.e. finding the same best parameter and training accuracy. The classification accuracy of APPROXEE is not significantly different from EE as most of the points fall very closely on the red diagonal line.

If $\mathcal{N}$ is sufficiently large, the best parameter estimated can be similar to learning with the full N time series but much faster. Then the estimated LOO-CV accuracy will be closer to the exact LOO-CV accuracy and thus does not significantly affect the final classification. However for very small $\mathcal{N}$, it is very likely that the NN classifier will not learn the best parameter. This is because the estimated LOO-CV accuracy can only be 0, 0.5 or 1 if $\mathcal{N} = 2$. This suggests that the classifiers in EE with approximate LOO-CV do not contribute much in classification accuracy.

The results from this experiment suggest that it is possible to ignore an elastic distance in EE without compromising on the classification accuracy. It is possible that EE does not need all 11 elastic distance measures to achieve such high classification accuracy. The exploration of this possibility is left for future work.

5.6 Conclusion

We propose the FASTEE algorithm – an extension of FASTWWS to the other elastic distance measures. New lower bounds have also been proposed for elastic distances without a previous bound, specifically WDTW, MSM and TWED. Lower bounds help in reducing the search space and are critical to the FASTEE algorithm. Our results showed that FASTEE is significantly faster than the standard implementation of EE and LBEE which uses lower bounds. To our surprise, we did not find any significant speed up in using a lower bound search to speed up the training time of EE. The main reasons are due to the inefficiency of computing the lower bound across all the parameters and that the tightness of the lower bounds degrades as the parameters change.

We also showed that it is possible to do an approximate LOO-CV process to select parameters for WDDTW, WDTW and TWED without significantly impacting accuracy. The approximation is done by learning from a subset $\mathcal{N}$ of the full training set $\mathcal{T}$ of size N. We showed that even a small $\mathcal{N}$ is sufficient to provide similar classification accuracy while being 40 times faster. Although the approximate version can be applied to all the other elastic distance measures, we leave it for future work. Our future work also includes the exploration of the possibility of using fewer classifiers in EE, indexing the training set and applying a contract time for training. The results presented in this work are important because if EE can be trained in a shorter time, then HIVE-COTE (the most accurate TSC algorithm) can also be made faster and more feasible.

Chapter 6

A New Framework and Methods to Lower Bound DTW

This chapter studies DTW lower bounds at various warping window sizes. The NN-DTW algorithm is at the core of state-of-the-art classification algorithms including the Ensembles of Elastic Distances (EE) [17] and Collection of Transformation-Based Ensembles (COTE) [21]. DTW's complexity makes NN-DTW highly computationally demanding. To combat this, lower bounds to DTW are used to minimize the number of times the expensive DTW need be computed during NN-DTW search. Effective lower bounds must balance 'time to calculate' vs 'tightness to DTW'. On the one hand, the tighter the bound the fewer the calls to the full DTW. On the other, calculating tighter bounds usually requires greater computation. Numerous lower bounds have been proposed. Different bounds provide different trade-offs between compute time and tightness. In this work, we present a new class of lower bounds that are tighter than the popular Keogh lower bound, while requiring similar computation time. Our new lower bounds take advantage of the DTW boundary condition, monotonicity and continuity constraints. In contrast to most existing bounds, they remain relatively tight even for large windows. A single parameter to these new lower bounds controls the speed-tightness trade-off. We demonstrate that these new lower bounds provide an exceptional balance between computation time and tightness for the NN-DTW time series classification task, resulting in greatly improved efficiency for NN-DTW lower bound search.

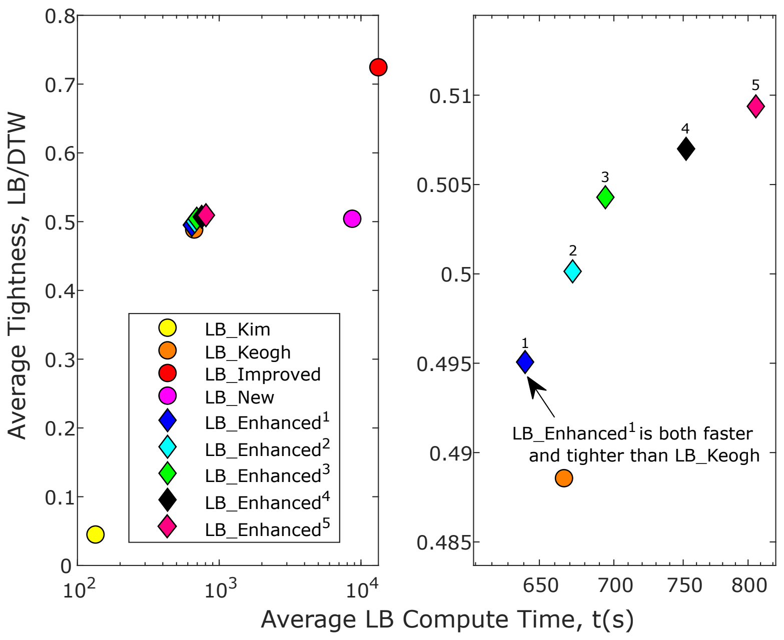


Figure 6.1: Tightness-Compute Time comparison of existing and our lower bounds at $w = 0.1 \cdot L$ over 250,000 time series pairs with $L = 256$ randomly sampled from the benchmark UCR time series archive [6]. Figure on the right shows the zoomed in plot. Our LB_ENHANCED^1 is faster and tighter than LB_KEOGH.

6.1 Introduction

Dynamic Time Warping (DTW) lower bounds play a key role in speeding up many forms of time series analytics [1, 26, 27, 32]. Several lower bounds have been proposed [27, 28, 29, 30, 31]. Each provides a different trade-off between compute time (speed) and tightness. Figure 6.1 illustrates this, plotting average tightness (LB$(S,T)/$ DTW(S,T)) against the average time to compute for alternative lower bounds. As shown in Figure 6.2, different bounds have different relative tightness at different window sizes.

In this chapter, we present a family of lower bounds, all of which are of $O(L)$ time complexity and are in practice tighter than LB_KEOGH. Two of these, LB_ENHANCED^1 and LB_ENHANCED^2, have very similar compute time to LB_KEOGH, while providing tighter bounds, meaning that their performance should dominate that of LB_KEOGH on any standard time series analysis task.

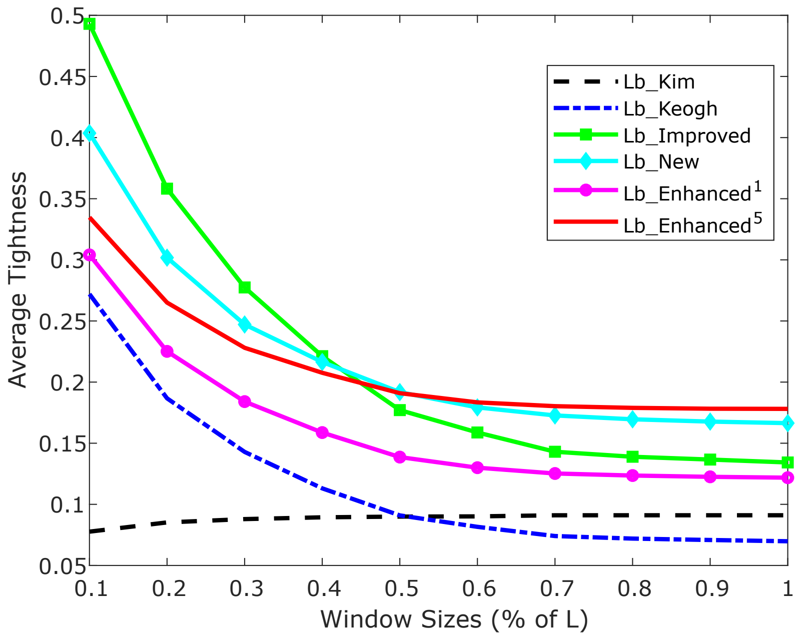


Figure 6.2: Tightness of different lower bounds at differing window sizes averaged across all UCR datasets [6]. Our very efficient LB_ENHANCED^5 is tighter than any alternative at large window sizes.

We focus on the application of lower bounds to DTW in Nearest Neighbor (NN-DTW) Time Series Classification (TSC). NN-DTW is in its own right a useful TSC algorithm and is a core component of the most accurate current TSC algorithms, COTE [21] and EE [17], which are ensembles of TSC algorithms. NN-DTW is thus at the core of TSC algorithms, but is extremely costly to compute [1, 3]. Given a training set with N time series and length L, a single classification with standard NN-DTW requires $O(N \cdot L^2)$ operations. Besides, NN-DTW is only competitive when used with a warping window, w learned from the training set [23]. Learning the best warping window is very time consuming as it requires the enumeration of numerous windows in the range of 0% to 100% of L and is extremely inefficient for large training sets [3].

There has been much research into speeding up NN-DTW, tackling either the N part [1] or the L^2 part of the complexity [27, 28, 29, 30, 31, 92]. A key strategy is to use lower bound search, which employs lower bounds on DTW to efficiently exclude nearest neighbor candidates without having to calculate NN-DTW [27, 28, 29, 30, 31]. We show that different speed-tightness trade-offs from different lower bounds prove most effective at speeding up NN-DTW for different window sizes.

Of the existing widely used lower bounds, LB_KIM [28] is the fastest, with constant time complexity with respect to window size. It is the loosest of the existing standard bounds for very small w, but its relative tightness increases as window size increases. For small window sizes, LB_KEOGH [27] provides an effective trade off between speed and tightness. However, as shown in Figure 6.2, it is sometimes even looser than LB_KIM at large window sizes. The more computationally intensive, LB_IMPROVED [31] provides a more productive trade-off for many of the larger window sizes. LB_NEW [30] does not provide a winning trade-off for this task at any window size.

The new DTW lower bound that we propose has the same complexity $O(L)$ as LB_KEOGH. Our new lower bound is parameterized, with a tightness parameter controlling a useful speed-tightness trade-off. At its lowest setting, LB_ENHANCED^1, it is uniformly tighter than LB_KEOGH. It replaces two calculations of the distance of a query point (the first and last) to a target LB_KEOGH envelope with two calculations of distances between a query and a target point. This may or may not be faster, depending whether the query point falls within the envelope, in which case LB_KEOGH does not perform a distance calculation. However, due to its greater tightness, this variant always supports faster NN-DTW. At V=$\{2,3,4,5\}$, our tighter LB_ENHANCED provide the greatest speed-up out of all standard DTW lower bounds for NN-DTW over a wide range of window sizes.

This chapter is organised as follows. In Section 6.2, we review relevant background and related work. Then we describe our proposed lower bound in Section 6.3. Section 6.4 presents an evaluation of our new lower bound in terms of its utility in TSC with NN-DTW. Lastly, we conclude our paper in Section 6.5.

6.2 Background and Related Work

We let $S = \langle S_1, \ldots, S_L \rangle$ and $T = \langle T_1, \ldots, T_L \rangle$ be a pair of time series S and T that we want to compare. Note that, for ease of exposition, we assume that the two series are of equal length, but the techniques trivially generalize to unequal length series. Section 2.2.2 and Section 2.4 give a detailed explanation for the DTW similarity measure and existing DTW lower bounds. This section will only provide the necessary background to understand the work presented in this chapter.

6.2.1 Dynamic Time Warping

DTW finds the global *alignment* of a time series pair, S and T, as illustrated in Figure 2.2a. The *warping path* of S and T is a sequence $\mathcal{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_P \rangle$ of *links*. Each link is a pair $\mathcal{A}_k = (i, j)$ indicating that S_i is aligned with T_j. $\mathcal{A}$ must obey the constraints outlined in Section 2.2.2. The cost of a warping path is minimised using dynamic programming by building a cost matrix $\mathcal{D}_{\text{DTW}}$, as illustrated in Figure 2.2b and computed using Equation 2.2. Then DTW(S, T) is calculated using Equation 6.1, where $\mathcal{A}_i^1$ is the first index in $\mathcal{A}_i$ and $\mathcal{A}_i^2$ the second.

$$\text{DTW}(S,T) = \sqrt{\mathcal{D}(L,L)} = \sqrt{\sum_{i=1}^{L} (S_{\mathcal{A}_i^1} - T_{\mathcal{A}_i^2})^2} \tag{6.1}$$

A global constraint on the warping path can be applied to DTW, such that $S(i)$ and $T(j)$ can only be aligned if they are within a window range, w. This constraint is known as the warping window, w (previously Sakoe-Chiba band) [45, 81] and we write this as DTW$_w(S, T)$. Figure 2.3 shows an example with warping window $w = 3$, where the alignment of S and T is constrained to be inside the gray band. More details can be found in Sections 2.2.2 and 4.2.2.

6.2.2 Existing DTW Lower Bounds

For the rest of the chapter, we will refer a lower bound as LB_⟨NAME⟩ and consider S as the query time series that is compared to T.

Kim Lower Bound (LB_KIM) [28] has $O(1)$ complexity and extracts four features – distances of the first, last, minimum, maximum points from the time series. Then the maximum of all four features is the lower bound for DTW. Equation 2.12 describes this lower bound.

Yi Lower Bound (LB_YI) [29] computes the sum of all the points in S that are larger than $\max(T)$ or smaller than $\min(T)$ as the lower bound for DTW. Refer to Equation 2.13 for the computation of this lower bound

Keogh Lower Bound (LB_KEOGH) [27] computes the upper UE^T and lower LE^T (Equation 2.14) envelopes. These are the upper and lower bounds on T within the window of each point in S. Then the lower bound is the sum of distances to the envelope of points in S that are outside the envelope of T as shown in Equation 2.15. LB_KEOGH is generally tighter than LB_KIM and LB_YI.

Improved Lower Bound (LB_IMPROVED) [31] computes a lower bound for DTW in two passes as described in Equation 2.17. First it computes LB_KEOGH and finds the projection of S on to the envelope of T – S' using Equation 2.16. Then it builds the envelope for S' and computes LB_KEOGH(T, S'). LB_IMPROVED is normally tighter than LB_KEOGH, but has higher computation overheads. It is usually used with an early abandon process to make it faster. If the first LB_KEOGH is sufficient to abandon the search, the expensive second pass is not performed.

New Lower Bound (LB_NEW) [30] takes advantage of the boundary and continuity conditions for a DTW warping path to create a tighter lower bound than LB_KEOGH. However, in general, LB_IMPROVED is tighter and is more effective at pruning NN candidates. Equation 2.18 describes this lower bound.

6.3 Proposed DTW Lower Bound

Our proposed lower bounds are based on the observation that the warping paths are very constrained at the start and the end of the series. Specifically, the boundary constraints require that the first link, $\mathcal{A}_1$, is $(1,1)$. The continuity and monotonicity constraints ensure that $\mathcal{A}_2 \in \{(1,2),(2,1),(2,2)\}$. If we continue this sequence of sets we get the *left bands*,

$$\mathcal{L}_i^w{=}\{(\max(1,i-w),i),(\max(1,i-w)+1,i),\ldots,(i,i),(i,i-1),\ldots,(i,\max(1,i-w)\}.$$

These are the alternating bands through the cost matrix shown in Figure 6.3. We can use these bands to define a lower bound on DTW as explained in Theorem 15.

Theorem 15. $\sum_{i=1}^{L}\min_{(j,k)\in\mathcal{L}_i^w}(S(j)-T(k))^2$ *is a lower bound on* $\mathrm{DTW}_w(S,T)$*.*

Proof. The continuity constraint requires that for all $1 \le i \le L$, any warping path $\mathcal{A}$ must include (i,p) and (q,i), for some $i-w \le p \le i+w$ and $i-w \le q \le i+w$. Either the indexes for both S and T reach i in the same pair and $p=q=i$, or one of the indexes must reach i before the other, and $p<q$ or $p>q$. If $p=q=i$, $(i,i)\in\mathcal{A}$. If $p<q$, $\mathcal{A}$ must contain one of $(i,\max(1,i-w)),\ldots(i,i-1)$. If $p>q$, $\mathcal{A}$ must contain one of $(\max(1,i-w),i),\ldots(i-1,i)$. Thus, $\mathcal{A}$ must contain (at least) one of $\mathcal{L}_i^w$. It follows that

$$\begin{aligned}
\mathrm{DTW}_w(S,T) &= \sum_{(j,k)\in\mathcal{A}}(S(j)-T(k))^2 \\
&\ge \sum_{i=1}^{L}\sum_{(j,k)\in\mathcal{L}_i^w\cap\mathcal{A}}(S(j)-T(k))^2 \\
&\ge \sum_{i=1}^{L}\min_{(j,k)\in\mathcal{L}_i^w}(S(j)-T(k))^2.
\end{aligned}$$

$\square$

		6	7	9	8	4	2	3	6	9	7	4	2
$\mathcal{L}_1^4$	0	**36**	49	81	64	16							
$\mathcal{L}_2^4$	3	**9**	16	36	25	1	1						
$\mathcal{L}_3^4$	4	**4**	9	25	16	**0**	4	1					
$\mathcal{L}_4^4$	6	**0**	1	9	4	4	16	9	0				
$\mathcal{L}_5^4$	5	1	4	16	9	1	9	4	1	16			
$\mathcal{L}_6^4$	4		9	25	16	**0**	4	1	4	25	9		
$\mathcal{L}_7^4$	3			36	25	1	1	**0**	9	36	16	1	
$\mathcal{L}_8^4$	4				16	**0**	4	1	4	25	9	**0**	4
$\mathcal{L}_9^4$	5					**1**	9	4	1	16	4	1	9
$\mathcal{L}_{10}^4$	4						4	**1**	4	25	9	0	4
$\mathcal{L}_{11}^4$	5							4	1	16	4	1	9
$\mathcal{L}_{12}^4$	6								**0**	9	1	4	16

Lower bound $(S, T) = 51$

Figure 6.3: The cost matrix for calculating a lower bound using *left* bands with $w = 4$. Alternating colors distinguish successive bands.

Figure 6.3 illustrates this lower bound in terms of the cost matrix with $w = 4$. The columns are elements of S and rows the elements of T. The elements in the matrix show the pairwise distances of each point in the time series pair S and T. Successive $\mathcal{L}_i^w$ are depicted in alternating colors. The minimum distance in each $\mathcal{L}_i^w$ is set in bold type. The sum of these minimums provides a lower bound on the DTW distance.

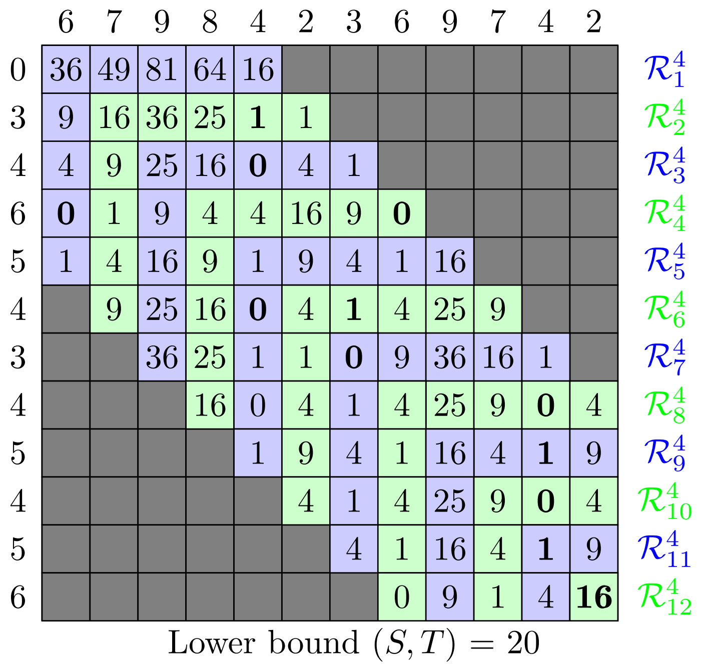
Working from the other end, the boundary constraints require that $\mathcal{A}_P$=(L, L). Continuity and monotonicity constraints ensure that at least one of the *right* band

$$\mathcal{R}_i^w = \{(\min(L, i+w), i), (\min(L, i+w)+1, i), \ldots, (i, i), (i, i-1), \ldots, (i, \min(L, i+w)\}$$

is in every warping path. Thus,

$$\text{DTW}_w(S, T) \geq \sum_{i=1}^{L} \min_{(j,k)\in\mathcal{R}_i^w} (S(j) - T(k))^2. \tag{6.2}$$

Figure 6.4 illustrates this lower bound in terms of the cost matrix. The proof of correctness of this bound is a trivial variant of the proof for Theorem 15.



	6	7	9	8	4	2	3	6	9	7	4	2	
0	36	49	81	64	16								$\mathcal{R}_1^4$
3	9	16	36	25	**1**	1							$\mathcal{R}_2^4$
4	4	9	25	16	**0**	4	1						$\mathcal{R}_3^4$
6	**0**	1	9	4	4	16	9	**0**					$\mathcal{R}_4^4$
5	1	4	16	9	1	9	4	1	16				$\mathcal{R}_5^4$
4		9	25	16	**0**	4	**1**	4	25	9			$\mathcal{R}_6^4$
3			36	25	1	1	**0**	9	36	16	1		$\mathcal{R}_7^4$
4				16	0	4	1	4	25	9	**0**	4	$\mathcal{R}_8^4$
5					1	9	4	1	16	4	**1**	9	$\mathcal{R}_9^4$
4						4	1	4	25	9	**0**	4	$\mathcal{R}_{10}^4$
5							4	1	16	4	**1**	9	$\mathcal{R}_{11}^4$
6								0	9	1	4	**16**	$\mathcal{R}_{12}^4$

Lower bound $(S, T) = 20$

Figure 6.4: The cost matrix for calculating a lower bound using *right* bands with $w = 4$. Alternating colors distinguish successive bands.

LB_KEOGH uses the minimum value from each band in Figure 6.5 so long as $S(i) > UE_w^T(i)$ or $S(i) < LE_w^T(i)$. When $LE_w^T(i) \leq S(i) \leq UE_w^T(i)$, the band is set in gray. For other bands, the minimum is set in bold. Then the sum over all minimum distances in non-gray bands gives LB_KEOGH. It is notable that the leftmost of the left bands and the rightmost of the right bands contain fewer distances than any of the LB_KEOGH bands. All things being equal, on average the minimum of a smaller set of distances should be greater than the minimum of a larger set. Further, because the number of distances in these few bands are small (and are invariant to window size), it is feasible to take the true minimum of the band, rather than taking an efficiently computed lower bound on the minimum, as does LB_KEOGH.

6.3.1 Enhanced Lower Bound

Based on these observations, our proposed lower bound exploits the tight leftmost and rightmost bands, but uses the LB_KEOGH bands in the centre section where the left and right bands are larger, and hence less tight and more expensive to compute. This is illustrated in Figure 6.6.

$$\begin{array}{c|cccccccccccc}
 & 6 & 7 & 9 & 8 & 4 & 2 & 3 & 6 & 9 & 7 & 4 & 2 \\
\hline
0 & 36 & 49 & 81 & 64 & 16 & & & & & & & \\
3 & 9 & 16 & 36 & 25 & 1 & \mathbf{1} & & & & & & \\
4 & 4 & 9 & 25 & 16 & 0 & 4 & 1 & & & & & \\
6 & 0 & \mathbf{1} & \mathbf{9} & \mathbf{4} & 4 & 16 & 9 & 0 & & & & \\
5 & 1 & 4 & 16 & 9 & 1 & 9 & 4 & 1 & 16 & & & \\
4 & & 9 & 25 & 16 & 0 & 4 & 1 & 4 & 25 & 9 & & \\
3 & & & 36 & 25 & 1 & 1 & 0 & 9 & 36 & 16 & 1 & \\
4 & & & & 16 & 0 & 4 & 1 & 4 & 25 & 9 & 0 & \mathbf{4} \\
5 & & & & & 1 & 9 & 4 & 1 & 16 & 4 & 1 & 9 \\
4 & & & & & & 4 & 1 & 4 & 25 & 9 & 0 & 4 \\
5 & & & & & & & 4 & 1 & 16 & 4 & 1 & 9 \\
6 & & & & & & & & 0 & \mathbf{9} & \mathbf{1} & 4 & 16
\end{array}$$

$$\text{LB\_K}{\scriptstyle\text{EOGH}}_4(S,T) = 29$$

Figure 6.5: The cost matrix for calculating LB_KEOGH$_4(S,T)$ with $w = 4$. Alternating colors distinguish successive bands. Where $LE_w^T(i) \leq S(i) \leq UE_w^T(i)$, the band is set in grey. For other bands, the minimum is set in bold.

$$\begin{array}{c|cccccccccccc}
 & 6 & 7 & 9 & 8 & 4 & 2 & 3 & 6 & 9 & 7 & 4 & 2 \\
\hline
0 & \mathbf{36} & 49 & 81 & 64 & 16 & & & & & & & \\
3 & \mathbf{9} & 16 & 36 & 25 & 1 & \mathbf{1} & & & & & & \\
4 & \mathbf{4} & 9 & 25 & 16 & 0 & 4 & 1 & & & & & \\
6 & \mathbf{0} & 1 & 9 & 4 & 4 & 16 & 9 & 0 & & & & \\
5 & 1 & 4 & 16 & 9 & 1 & 9 & 4 & 1 & 16 & & & \\
4 & & 9 & 25 & 16 & 0 & 4 & 1 & 4 & 25 & 9 & & \\
3 & & & 36 & 25 & 1 & 1 & 0 & 9 & 36 & 16 & 1 & \\
4 & & & & 16 & 0 & 4 & 1 & 4 & 25 & 9 & 0 & 4 \\
5 & & & & & 1 & 9 & 4 & 1 & 16 & 4 & \mathbf{1} & 9 \\
4 & & & & & & 4 & 1 & 4 & 25 & 9 & \mathbf{0} & 4 \\
5 & & & & & & & 4 & 1 & 16 & 4 & \mathbf{1} & 9 \\
6 & & & & & & & & 0 & 9 & 1 & 4 & \mathbf{16}
\end{array}$$

$$\text{LB\_E}{\scriptstyle\text{NHANCED}}_4^4(S,T) = 68$$

Figure 6.6: Cost matrix for calculating LB_ENHANCED$_4^4(S,T)$ with $V = 4$ and $w = 4$. Alternating colors distinguish successive bands. Where $LE_w^T(i) \leq S(i) \leq UE_w^T(i)$, the band is set in grey. For other bands, the minimum is set in bold.

LB_ENHANCED is parametrized by a tightness parameter V, $1 \leq V \leq L/2$, that specifies how many *left* and *right* bands are utilized. This controls the speed-tightness trade-off. Smaller Vs require less computation, but usually result in looser bounds, while higher values require more computation, but usually provide tighter bounds, as illustrated in Figure 6.1. LB_ENHANCED is defined as follows

$$\text{LB\_ENHANCED}_w^V(S,T) = \sum_{i=1}^{L} \begin{cases} \min_{(j,k)\in\mathcal{L}_i^w}(S(j)-T(k))^2 & \text{if } i \leq V \\ \min_{(j,k)\in\mathcal{R}_i^w}(S(j)-T(k))^2 & \text{if } i > L-V \\ (S(i)-UE_w^T(i))^2 & \text{if } S(i) > UE_w^T(i) \\ (S(i)-LE_w^T(i))^2 & \text{if } S(i) < LE_w^T(i) \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

where $UE_w^T(i)$ and $LE_w^T(i)$ are defined in Equation 2.14.

Theorem 16. *For any two time series S and T of length L, for any warping window, $w \leq L$, and for any integer value $V \leq L/2$, the following inequality holds:* LB_ENHANCED$_w^V(S,T) \leq$ DTW$_w(S,T)$

Proof. From the proof for Theorem 15, for every $1 \leq i \leq L$, $\mathcal{A}$ must contain (at least) one of $\mathcal{L}_i^w$ and with trivial recasting this also establishes that for every $1 \leq i \leq L$, $\mathcal{A}$ must contain (at least) one of $\mathcal{R}_i^w$.

To address the contribution of the LB_KEOGH inspired bridge between the $\mathcal{L}$s and $\mathcal{R}$s, we introduce the notion of a vertical band $\mathcal{V}_i$ for element $S(i)$. $\mathcal{V}_i = \{(i,j) : \max(1, i-w) \leq j \leq \min(L, i+w)\}$ is the set of pairs containing $S(i)$ that may appear in a warping path. Note that $\mathcal{L}_1^w, \ldots, \mathcal{L}_V^w$, $\mathcal{V}_{V+1}, \ldots, \mathcal{V}_{L-V}$ and $\mathcal{R}_{L-V+1}^w, \ldots, \mathcal{R}_L^w$ are all mutually exclusive. None of these sets intersects any of the others. It follows

$$\begin{aligned}
\text{DTW}_w(S,T) &= \sum_{(j,k)\in\mathcal{A}} (S(j)-T(k))^2 \\
&\geq \sum_{i=1}^{V} \sum_{(j,k)\in\mathcal{L}_i^w\cap\mathcal{A}} (S(j)-T(k))^2 + \sum_{i=V+1}^{L-V} \sum_{(j,k)\in\mathcal{V}_i\cap\mathcal{A}} (S(j)-T(k))^2 + \\
&\quad \sum_{i=L-V+1}^{L} \sum_{(j,k)\in\mathcal{R}_i^w\cap\mathcal{A}} (S(j)-T(k))^2 \\
&\geq \sum_{i=1}^{V} \min_{(j,k)\in\mathcal{L}_i^w} (S(j)-T(k))^2 + \sum_{i=V+1}^{L-V} \min_{(j,k)\in\mathcal{V}_i} (S(j)-T(k))^2 + \\
&\quad \sum_{i=L-V+1}^{L} \min_{(j,k)\in\mathcal{R}_i^w} (S(j)-T(k))^2 \\
&\geq \sum_{i=1}^{L} \begin{cases}
\min_{(j,k)\in\mathcal{L}_i^w} (S(j)-T(k))^2 & \text{if } i \leq V \\
\min_{(j,k)\in\mathcal{R}_i^w} (S(j)-T(k))^2 & \text{if } i > L-V \\
(S(i)-UE_w^T(i))^2 & \text{if } S(i) > UE_w^T(i) \\
(S(i)-LE_w^T(i))^2 & \text{if } S(i) < LE_w^T(i) \\
0 & \text{otherwise}
\end{cases}
\end{aligned}$$

□

To illustrate our approach, we present the differences between LB_KEOGH and LB_ENHANCED^4_w with respect to S in Figures 6.5 and 6.6 respectively. In Figure 6.5, the i^{th} column represents $\mathcal{V}_i$, the possible pairs for $S(i)$. The columns are greyed out if $LE_w^T(i) \leq S(i) \leq UE_w^T(i)$, showing that they do not contribute to LB_KEOGH. For the remaining columns, the numbers in bold are the minimum distance of $S(i)$ to a $T(j)$ within $S(i)$'s window, either $LE_w^T(i) - S(i)$ or $S(i) - UE_w^T(i)$. The lower bound is the sum of these values. In Figure 6.6, alternate bands are set in alternating colors. The columns are greyed out if $V < i \leq L - V$ and $LE_w^T(i) \leq S(i) \leq UE_w^T(i)$, showing that they do not contribute to LB_ENHANCED^4_w. For the remaining columns, the numbers in bold are the minimum distance of $S(i)$ to $T(j)$ within the band. The lower bound is the sum of these values. These figures clearly show the differences of LB_KEOGH and LB_ENHANCED^4_w, where we take advantage of the tighter *left* and *right* bands.

Algorithm 17: LB_ENHANCED$(S, T, UE_w^T, LE_w^T, w, V, \mathtt{D})$

```
Input: $S$: Query time series
Input: $T$: Candidate time series
Input: $UE_w^T$: Upper Envelope for $T$
Input: $LE_w^T$: Lower Envelope for $T$
Input: $w$: Warping window
Input: $V$: Speed-Tightness parameter
Input: D: Current distance to NN
$res \leftarrow (S(1) - T(1))^2 + (S(L) - T(L))^2$
$nBands \leftarrow \min(L/2, V)$
// Do $\mathcal{L}$, $\mathcal{R}$ bands
for $i \leftarrow 2$ to $nBands$ do
    $min_{\mathcal{L}} \leftarrow \delta(S_i, T_i)$
    $min_{\mathcal{R}} \leftarrow \delta(S_{L-i+1}, T_{L-i+1})$
    for $j \leftarrow \max(1, i - w)$ to $i - 1$ do
        $min_{\mathcal{L}} \leftarrow \min(min_{\mathcal{L}}, (S(i) - T(j))^2)$
        $min_{\mathcal{L}} \leftarrow \min(min_{\mathcal{L}}, (S(j) - T(i))^2)$
        $min_{\mathcal{R}} \leftarrow \min(min_{\mathcal{R}}, (S(L-i+1) - T(L-j+1))^2)$
        $min_{\mathcal{R}} \leftarrow \min(min_{\mathcal{R}}, (S(L-j+1) - T(L-i+1))^2)$
    end
    $res \leftarrow res + min_{\mathcal{L}} + min_{\mathcal{R}}$
end
if $res \geq \mathtt{D}$ then
    return $\infty$
end
// Do LB_KEOGH
for $i \leftarrow nBands + 1$ to $L - nBands$ do
    if $S(i) > UE_w^T(i)$ then
        $res \leftarrow res + (S(i) - UE_w^T(i))^2$
    end
    else if $S(i) < LE_w^T(i)$ then
        $res \leftarrow res + (S(i) - LE_w^T(i))^2$
    end
end
return $res$
```

We apply a simple technique to make LB_ENHANCED more efficient and faster. In the naïve version, LB_ENHANCED has to compute the minimum distances of $\mathcal{L}$s and $\mathcal{R}$s. Usually, these computations are very fast as $\mathcal{L}$s and $\mathcal{R}$s are much smaller compared to $\mathcal{V}$, especially when L is long. To optimise LB_ENHANCED, we can first sum the minimum distances for the $\mathcal{L}$s and $\mathcal{R}$s. Then, if this sum is larger than the current distance to the nearest neighbour, D, we can abort the computation for $\mathcal{V}$.

Algorithm 17 describes our proposed lower bound. First, we compute the distance of the first and last points as set by the boundary condition. In line 2, we define the number of $\mathcal{L}$ and $\mathcal{R}$ bands to utilise. This number depends on the warping window, w, as we can only consider the points within w no matter how big V is. Line 3 to 11 computes the sum of the minimum distances for $\mathcal{L}$ and $\mathcal{R}$. If the sum is larger than the current distance to the nearest neighbour, we abort the computation in line 12. Otherwise, we do standard LB_KEOGH in lines 13 to 15.

6.4 Empirical Evaluation

Our experiments are divided into two parts, we first study the effect of the tightness parameter V in LB_ENHANCED. Then we show how well LB_ENHANCED can speed up NN-DTW compared to the other lower bounds. We used all the 85 UCR benchmark datasets [6] and the given train/test splits. The relative performance of different lower bounds varies greatly with differing window sizes. In consequence we conduct experiments across a variety of different window sizes, drawn from two sets of values. The set $w=\{1, \dots, 10\}$, spans the best warping windows for most of the UCR benchmark datasets. The set, $w=\{0.1 \cdot L, 0.2 \cdot L, \dots, L\}$ shows that using NN-DTW with LB_ENHANCED is always faster across the broad spectrum of all possible windows.

A NN-DTW with lower bound search can be further sped up by ordering the candidates in the training set based on a proxy for their relative distances to the query, such as a lower bound on that distance [1]. However, using a lower bound to order the candidates would unfairly advantage whichever bound was selected. Hence we order the training set by their Euclidean distance to the query time series (an upper bound on their true distance) and start with the candidate that gives the smallest Euclidean distance. Note that our LB_ENHANCED has even greater advantage if random order is employed.

All experiments were optimised and implemented in Java 8 and conducted on a 64-bit Linux AMD Opteron 62xx Class CPU @2.4GHz machine with 32GB RAM and 4 CPUs. Our source code are open-source at https://github.com/ChangWeiTan/LbEnhanced and the full results at http://bit.ly/SDM19.

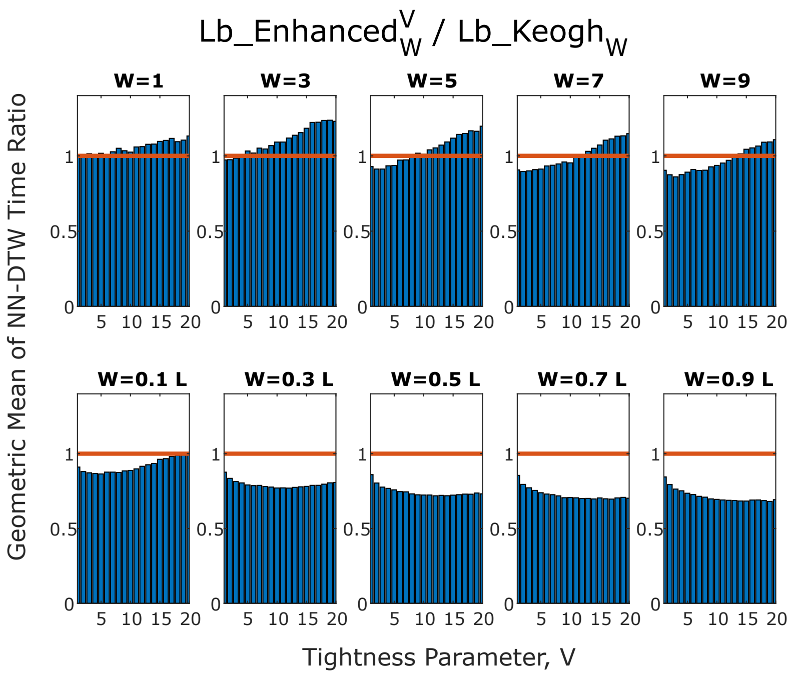


Figure 6.7: Geometric mean of NN-DTW classification time with LB_ENHANCED of different tightness parameter V normalised by LB_KEOGH. Smaller ratio means faster, and values below 1 (under the red line) indicate LB_ENHANCED is faster than LB_KEOGH.

6.4.1 How to Choose the Right Tightness Parameter for LB_ENHANCED?

Recall that our LB_ENHANCED is parametrized by a tightness parameter V that specifies the number of bands to be used. This parameter controls the speed tightness trade-off. Higher values require more computations but usually gives tighter bounds. We conducted a simple experiment by recording the classification time of NN-DTW with LB_KEOGH as the baseline and LB_ENHANCED with different tightness parameter in the range of $V=\{1,\ldots,20\}$. Note that all the required envelopes have been pre-computed at training time and the time is not included in the classification time. Then the classification time of NN-DTW with LB_ENHANCED is normalised by LB_KEOGH. Finally the geometric mean is computed over all 85 datasets.

The results are presented in Figure 6.7 where we show the performance for a subset of windows. The x axis shows the different Vs, the y axis shows the geometric mean of the normalised time. Ratios below 1 (under the red line) means that LB_ENHANCED is faster than LB_KEOGH and smaller ratio means faster LB_ENHANCED.

These plots show that the optimal value of V increases with w. At $w = 1$, only $V = 1$ and $V = 2$ outperform LB_KEOGH. At $w = 3$, all $V < 5$ prove to speed up NN-DTW more than LB_KEOGH, and subsequently $V < 10$ for $w = 5$ and $V < 15$ for $w = 7$ and $w = 9$. At larger window sizes, all $V < 20$ are more effective at speeding up NN-DTW than LB_KEOGH. For $w = 0.1 \times L$, $V = 5$ proves to be most effective. For $w \geq 0.3 \times L$, there are a wide range of values of V with very similar performance. One reason for this is that window size is not the only factor that affects the optimal value of V. Series length and the amount of variance in the sequence prefixes and suffixes are further relevant factors. $V = 5$ provides strong performance across a wide range of window sizes. In consequence, in the next section we choose $V = 5$ and compare the performance of LB_ENHANCED^5 to other lower bounds.

6.4.2 Speeding Up NN-DTW with LB_ENHANCED

We compare our proposed lower bounds against four key existing alternatives, in total 5 lower bounds:

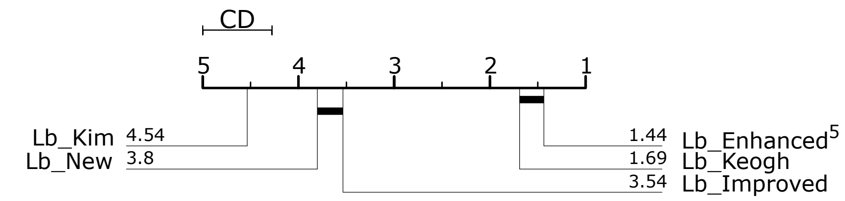
- **LB_KIM:** The original LB_KIM proposed in [28] is very loose and incomparable to the other lower bounds. To make it tighter and comparable, instead of the maximum, we take the sum of all the four features without repetitions.
- **LB_KEOGH:** We use the original implementation of LB_KEOGH proposed in [27].
- **LB_IMPROVED:** We use the original implementation of LB_IMPROVED and the optimised algorithm to compute the projection envelopes proposed in [31].
- **LB_NEW:** We use the original implementation of LB_NEW proposed in [30].
- **LB_ENHANCED:** We use LB_ENHANCED^5, selecting $V = 5$ as it provides reasonable speed up across a wide range of window sizes.

Note that LB_YI was omitted because it is similar to LB_KEOGH when $w = L$. Similar to before, we record the classification time of NN-DTW with the various lower bounds. For each dataset we determine the rank of each bound, the fastest receiving rank 1 and the slowest rank 5. Figure 6.8 shows the critical difference diagram comparing the classification time ranks of each lower bound for $w = \{3, 6, 10, 0.1 \cdot L, 0.5 \cdot L\}$. The results for the remaining of the windows can be found in Appendix B and http://bit.ly/SDM19. Each plot shows the average rank of each lower bound (the number next to the name). Where the ranks are not significantly different, their corresponding lines are connected by a solid line. Thus, for $w = 10$, LB_ENHANCED^5 has significantly lower average rank than any other bound and the average ranks of LB_KEOGH and LB_IMPROVED do not differ significantly but are significantly lower than those of LB_NEW and LB_KIM; which in turn do not differ significantly from one another.

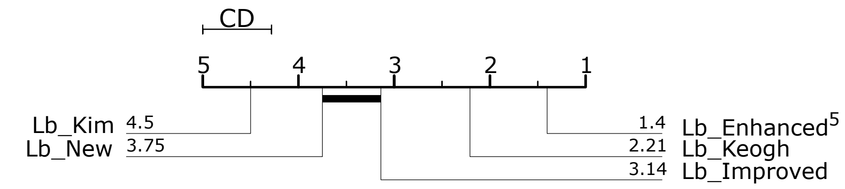
Our LB_ENHANCED^5 has the best average rank of all the lower bounds at all window sizes, significantly so at $w = 6$ to 10 and $0.5 \cdot L$ to L. For smaller windows, LB_KEOGH is lowest ranked of the remaining bounds. For larger windows, the tighter, but more computationally demanding LB_IMPROVED comes to the fore.

We further extend our analysis by computing the speed up gained from LB_ENHANCED^5 relative to the other lower bounds. We compute the NN-DTW time ratio of all the other lower bounds to LB_ENHANCED^5 for $w = \{1, \ldots 10\}$ and present the geometric mean (average) in Figure 6.9. The results show that LB_ENHANCED^5 is consistently faster than all the other lower bounds.

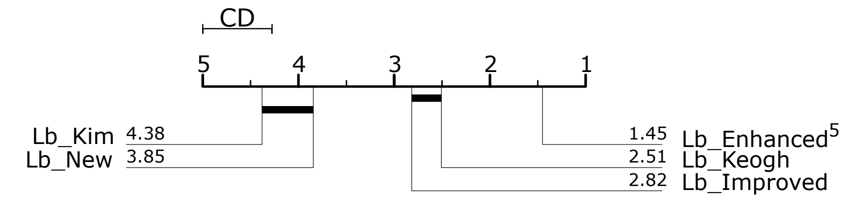
It might be thought that our experimental comparison has unfairly penalized LB_KEOGH and LB_NEW relative to LB_IMPROVED and LB_ENHANCED, as only the latter use a form of early abandoning [32]. However, LB_IMPROVED starts with LB_KEOGH and LB_ENHANCED uses LB_KEOGH for most of the sequence. Hence, each of these could benefit as much as would LB_KEOGH from the adoption of early abandoning in the LB_KEOGH process.



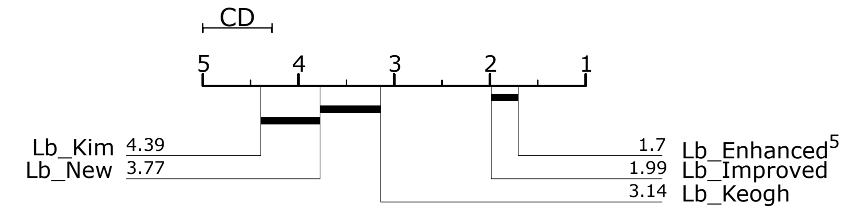
(a) $w = 3$



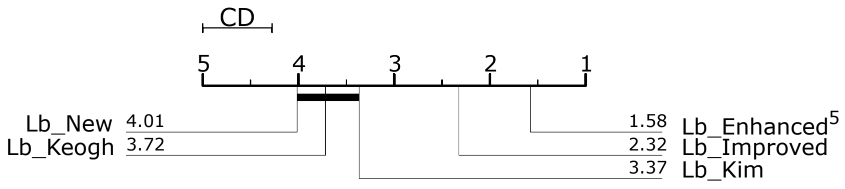
(b) $w = 6$



(c) $w = 10$



(d) $w = 0.1 \cdot L$



(e) $w = 0.5 \cdot L$

Figure 6.8: Ranking of all lower bounds in terms of NN-DTW classification time.

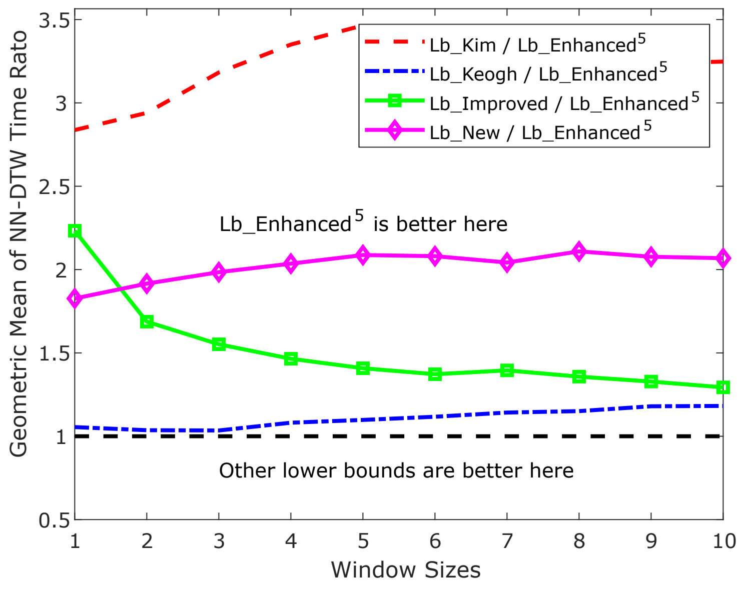


Figure 6.9: Geometric mean (average) ratio of classification time for major existing lower bounds to our proposed new lower bound LB_ENHANCED^5 across all benchmark datasets [6] at $w = \{1, \dots 10\}$.

6.5 Conclusion and Future Work

In conclusion, we proposed LB_ENHANCED, a new lower bound for DTW. The speed-tightness trade-off of LB_ENHANCED results in faster lower bound search for NN-DTW than any of the previous established bounds at all window sizes. We expect it to be similarly effective at a wide range of nearest neighbour retrieval tasks under DTW. We showed that choosing a small tightness parameter V is sufficient to effectively speed up NN-DTW. Although it is possible to learn the best V for a dataset (which will be future work), our results show that when $V = 5$, NN-DTW with LB_ENHANCED^5 is faster and more efficient than with the existing lower bounds for all warping window sizes across 85 benchmark datasets.

In addition, there is potential to replace LB_KEOGH by LB_IMPROVED within LB_ENHANCED. This would increase the computation time, but the strong performance of LB_IMPROVED suggests it should result in a powerful trade-off between time and tightness especially at larger windows. Finally, since LB_KEOGH is not symmetric with respect to S and T, LB_ENHANCED is not symmetric too. Thus, $\max($LB_ENHANCED$(S,T),$ LB_ENHANCED$(T,S))$ is also a useful bound. Our proposed lower bound could also be cascaded [32], which may further improve pruning efficiency.

Chapter 7

Tamping Effectiveness Prediction With Time Series Classification

In collaboration with the Institute of Railway Technology (IRT) at Monash University, this chapter shows the application of TSC in real-world problem – improving railway track maintenance. Railway maintenance planning is critical in maintaining track assets. Tamping is a common railway maintenance procedure and is often used when geometrical issues are first identified. Tamping repacks ballast particles under sleepers to restore the correct geometrical position of ballasted tracks. However, historical data shows that tamping is not always effective in restoring track to a satisfactory condition. Furthermore, ineffective, or unnecessary tamping tends to reduce the lifetime of existing track.

An intuitive way of preventing ineffective tamping is to predict the likely tamping effectiveness. This work aims to predict the likely tamping effectiveness ahead of time using supervised machine learning techniques that predict an outcome using labelled training data. In this case, the training database consists of multivariate sensor data from the Instrumented Revenue Vehicles (IRVs). The data between the previous and current tamping dates are used. This forms a time series database labelled with the tamping effectiveness of each track location based on the responses recorded from the IRVs before and after tamping. The labelled time series database is then used to train a time series classifier for prediction.

This work uses the state-of-the-art time series classification algorithm, k-Nearest Neighbour (k-NN) extended to the case of multivariate time series. k-NN is a non-parametric algorithm that does not make assumptions on the underlying model of the training data. With a sufficiently large training database, non-parametric algorithms can outperform parametric algorithms. Using k-NN, the tamping effectiveness of a potential tamping location that is not in the training database, or locations in the next tamping cycle, is predicted using the expected tamping effectiveness from a location in the training database that is the most similar to the target. This allows the algorithm to effectively to identify locations where tamping is likely to be ineffective. This work achieves high accuracy in the prediction of tamping effectiveness even at 12 weeks before tamping. It is hoped that the methodology will help in assisting decision making for maintenance planning activities.

7.1 Introduction

Irregular track geometry can incite undesirable vehicle dynamic response modes that increase track loading, reduce infrastructure component life, and increase the risk of vehicle derailment. It is therefore necessary to maintain track geometry to prevent the cyclic loading and vibrations from network traffic leading to the settlement of ballast formations, that can further exacerbate the degradation of track geometry via the establishment of a feedback process. Track geometry issues commonly trigger a range of track maintenance operations and renewals that have a range of costs associated with them.

To both control and minimize these costs, track operations and maintenance procedures need to be optimized. However, before this can be achieved, it is first necessary to understand how the track degrades and the consequences of the degradation in terms of cost and safety. With this information, railway operators are then in a better position to estimate the timing of inspections, maintenance, and renewals.

There are many types of track maintenance and renewal activities with the common one being tamping. Tamping is often used when geometrical issues are first identified. It repacks ballast particles under sleepers to restore the track to its correct geometrical position. Although tamping is a common maintenance procedure, historical data shows that it is not always effective and appropriate in restoring the track to a desired track geometry. Moreover, ineffective, or unnecessary tamping will reduce the lifetime of existing track, which is counter to the purpose of track maintenance.

An intuitive way of preventing ineffective tamping is to predict the effectiveness of tamping, ahead of time. Being able to predict the likely tamping effectiveness in advance enables planners to focus on the most appropriate alternative procedures, which in turn reduces the operational and maintenance costs. Recent work [122] shows that it is possible to evaluate the tamping effectiveness using continuous measured performance data collected from Instrumented Revenue Vehicles.

This study proposes a method to predict the effectiveness of tamping by extending the state-of-the-art time series classification algorithm, k-Nearest Neighbour (k-NN) to multivariate time series data.

7.2 Background and Motivation

This section gives a background in railway maintenance and the importance of it.

7.2.1 Tamping Maintenance Procedure

Tamping is a common maintenance procedure for restoring the geometrical position of track caused by ballast settlement. Tamping increases the supportive effect around and under the sleepers by compacting the ballast. It is usually done with a tamping machine where tamping tynes are inserted into the ballast on either side of the sleeper. The sleeper is then raised to the target level creating a space under the sleeper. The cavities underneath the sleeper are then filled by adjusting the ballast using tamping tools. The lining tool of the tamping machine adjusts the position of both rails so that the track is straightened and parallel. A typical tamping sequence is shown in Figure 7.1.

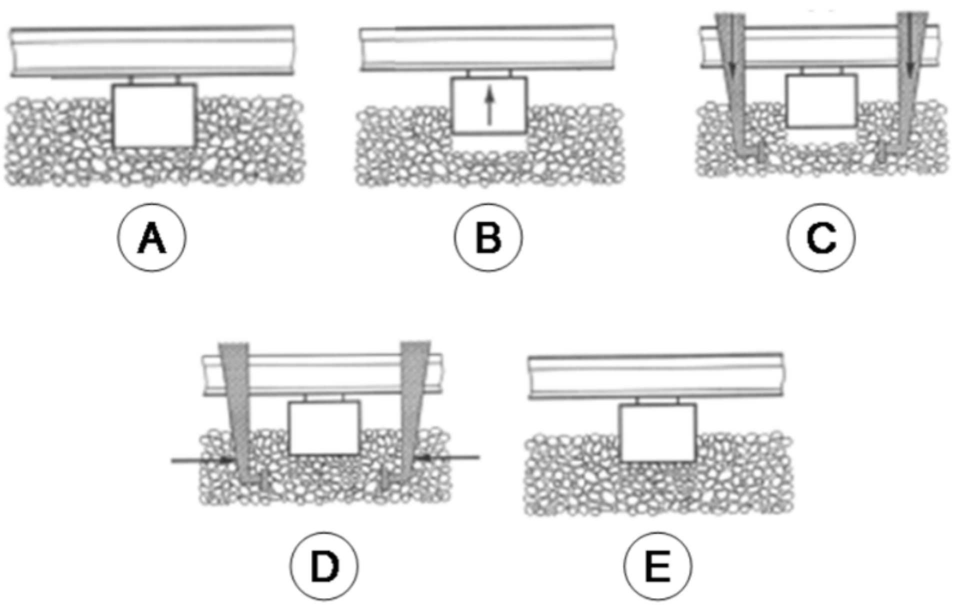


Figure 7.1: Standard tamping procedure [19].

Geometric irregularities in track that may benefit from tamping can be identified in several ways which include: assessment during maintenance work, visual inspections by track inspectors, reports from train drivers, track geometry car measurements or by monitoring for wagon-track dynamic activity.

7.2.2 Instrumented Revenue Vehicles (IRV)

Instrumented Revenue Vehicles (IRVs) are continuous monitoring systems that use the measured wagon dynamic activity to infer information about the underlying track condition [12]. The IRV system is a flexible fully automated integrated condition monitoring platform, which can be installed on the rolling stock used for normal revenue and it provides continuous feedback on both rail condition and train operation [11, 13].

IRVs have several key advantages over other maintenance inspection methods. Firstly, IRVs reduce the need for track downtime as they measure the condition of the system as part of normal operations. Secondly, the measurement of the dynamic response of IRVs provides a direct indication of the loads being imposed on the rail network, which are often not as clearly defined when reviewing data collected by other methods such as track recording vehicles and visual inspections. This study will demonstrate that the data from IRVs can also be used to predict the likely tamping effectiveness.

7.2.3 Motivation

An assessment of the data shows that tamping is not always effective in restoring tracks to their nominal condition. It was also observed that ineffective tamping tends to reduce the lifetime of existing track and increase the risk of failure.

To the best of our knowledge, there has been little to no prior research into the prediction of tamping effectiveness. Most research in railway track maintenance has tended to focus on estimating train-track dynamics [123], predicting track degradation [124, 125] and maintenance planning [126].

Tamping effectiveness prediction is a challenging problem due to the many complexities involved, such as local geology, soil quality, weather, age of the tracks, time and more. It is common for tracks sections in different locations that have nominally experienced the same loading conditions to respond quite differently to tamping. Tamping effectiveness prediction is therefore important in maintenance planning for three reasons:

1. It can minimize maintenance costs and time by assisting in the selection of more appropriate maintenance procedures.
2. It can reduce unplanned track possessions, by minimizing unnecessary tamping.
3. It avoids the cost of failure recovery.

Given the above three reasons, we are motivated to develop a system that can predict tamping effectiveness and which in turn will assist in decision making for track maintenance planning activities.

7.3 Methodology

Machine learning techniques and time series analysis has been applied to analyse the track geometry data for tamping effectiveness prediction. This section defines the terms required to understand the work in this chapter and describes the methodology used.

7.3.1 Time Series

Time series is defined as a sequence of observations ordered in time, where time is the independent variable. In this study, a univariate, one dimensional time series S is defined in Equation 7.1 as a sequence of real numbers paired with a discrete timestamp in the form of:

$$S = \{(S(1), t(1)), (S(2), t(2)), \ldots, (S(L), t(L))\} \tag{7.1}$$

where L is the number of data points.

Then, a multivariate time series (MTS) $\mathtt{S}$ as defined in Equation 7.2 is a finite sequence of univariate time series, S_j in a D-dimensional Euclidean space:

$$\mathtt{S} = \{S_1, S_2, \ldots, S_D\} \tag{7.2}$$

where D is the dimensionality of the multi-dimension series $\mathtt{S}$.

7.3.2 Dynamic Time Warping

In this work, Dynamic time warping (DTW) is used to compare a pair of time series. Time series in real-world applications are often shifted in time. Some have different lengths. For example, a speech signal of the same word can be spoken at a different rate, which will be shifted in time. The typical Euclidean distance (ED) metric will not be able to handle this. DTW on the other hand, is robust to time shifts. It aligns time series that are shifted in time. Figure 2.1 and 2.2a show a visual comparison of ED and DTW.

Section 2.2.2 describes the computation of DTW distance between two univariate time series S and T. Following the definition for multivariate time series (MTS) in Equation 7.2, the DTW distance between MTS $\mathtt{S}$ and $\mathtt{T}$ can be defined as the sum of all the DTW distances across all the D dimensions as shown in Equation 7.3 [127].

$$\text{DTW}^D(\mathtt{S}, \mathtt{T}) = \sum_{i+1}^{D} \text{DTW}(S_i, T_i) \tag{7.3}$$

7.3.3 Nearest Neighbour (NN) Algorithm

Time series classification is an important application of time series analysis and supervised machine learning. It maps an unlabelled time series (the target) to a label by training a classifier on a labelled example training database.

There are many classification algorithms such as Naïve Bayes [128], and Classification Trees [49, 50] that works based on features extracted from the time series. However, in most applications, the extracted features change over time, which makes it “incorrect” to classify by observing only the static features.

The NN algorithm on the other hand has shown to perform best on most datasets. It is the state-of-the-art time series classification algorithm that has been applied in many applications such as speech recognition [81], digit recognition [129], text mining [130] and gene expression classification [131]. With sufficiently large data, the NN algorithm usually outperforms the other algorithms.

The NN algorithm is one of the simplest yet highly effective classification algorithms. It is a non-parametric, lazy learning algorithm that does not necessary require any training phase on the training dataset [109]. It labels the unlabelled time series by searching for the most similar (nearest) time series from a database based on a similarity measure.

Often, finding more neighbours is of interest. More neighbours help in improving the confidence and reducing noise of the classified result. This is commonly known as the k nearest neighbours (k-NN) classifier. A k-NN classifier finds k closest neighbours of the unlabelled time series in the training database and returns the dominating class as the class for the unlabelled time series [109]. k is often chosen as an odd number to be able to return the dominating class (majority vote) in the k nearest neighbours set.

7.4 Experimental Design

In this study, we believe that there exist some patterns in the data that discriminate between the different tamping effectiveness. For instance, a consistent high response might indicate that tamping will not be effective. However, some locations might observe a different pattern even if they have the same tamping effectiveness. Hence, the NN-DTW algorithm that finds similar time series will be the first approach.

Our experiments are conducted using the Matlab R2015a software and are divided into two parts:

A. First, k-NN-DTW is compared with the different classification algorithms such as Naïve Bayes and Classification Tree to show that it is the most appropriate classifier for this task. The optimal k value will also be investigated

B. Then a study was performed to show the prediction performance of tamping effectiveness at different leading intervals by truncating the time series to x number of days before tamping. Typically, the earlier the tamping effectiveness can be predicted, the better it will be for the purposes of maintenance planning.

7.4.1 Data Acquisition and Processing

This study uses the dynamic response data as measured by the IRV at four locations, one on each side frame. This is known as Spring Nest Deflection (SND). The raw data is then typically processed after each trip or journey to map the measured responses to locations on track. Using multiple trips, the data is pre-processed to form a 4-dimensional time series for each track location, where each dimension (known as LP) represents the responses for each side frame. In this case, six months of trip data between February to July 2016 was used. The responses were then aggregated by taking the maximum value over 50 metres blocks of track and sorted in increasing time.

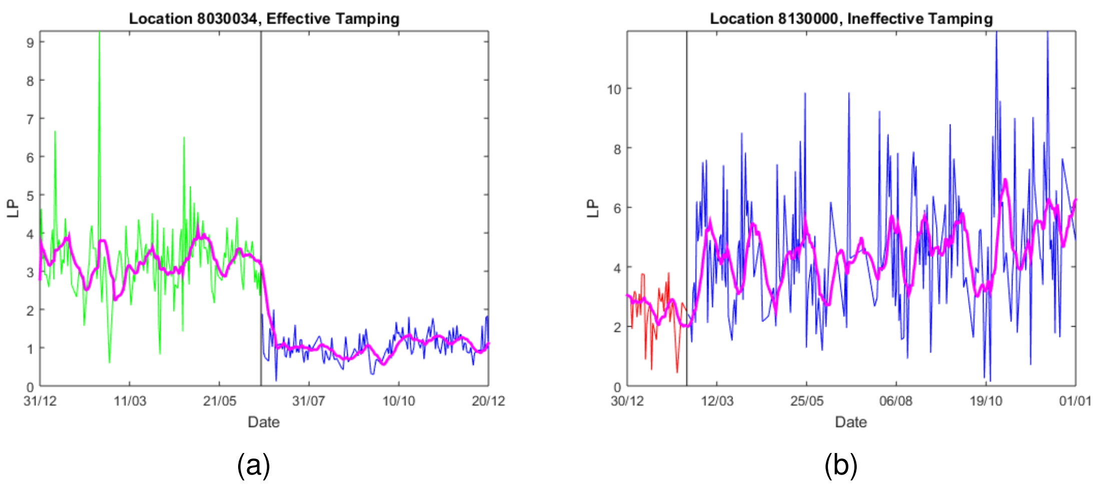


Figure 7.2: Green series represents (a) effective tamping. Red series represents (b) ineffective tamping. Blue series represents data after tamp. Magenta series is the 12 days moving average. Black line represents the tamping date.

IRV field recordings incorporate variability in both the operating environment as well as other uncontrollable operational factors. It should be noted that different locations were subjected to varying number of IRV trips. Those locations with fewer trips (less than 30 trips) forming shorter time series were not considered. The reason being, that shorter time series were considered to not have enough information about the track. Missing data from signal loss or faulty sensors was interpolated linearly. Locations with more than 50% of the data missing on one channel were also not considered.

Each of the locations are labelled with their respective tamping effectiveness. The tamping effectiveness label is calculated using the ratio of the average data after tamping compared to that before. In this study, only effective and ineffective tamping are considered. Note that it is possible to have more levels of tamping effectiveness. A ratio that is less than 1 is labelled as effective tamping, while a ratio that is more than 1 is ineffective tamping. Figure 7.2a shows that tamping is effective in reducing the SND measurements but not the case in Figure 7.2b.

Locations without tamping records are not considered as well. Lastly, the labelled data are re-sampled to days by taking the average response in that day. This makes every point comparable. Days with missing data are linearly interpolated as well. In this study, this forms a 4-dimensional time series training database with 985 samples. Table 7.1 shows the properties of the time series dataset used in this study.

IRV's SND Recording	
Dimensions	4 (LP$_1$, LP$_2$, LP$_3$, LP$_4$)
Length	37-184 days
Size of Dataset	985 track locations
Number of classes	2 (Effective and Ineffective)

Table 7.1: Properties of IRV time series dataset

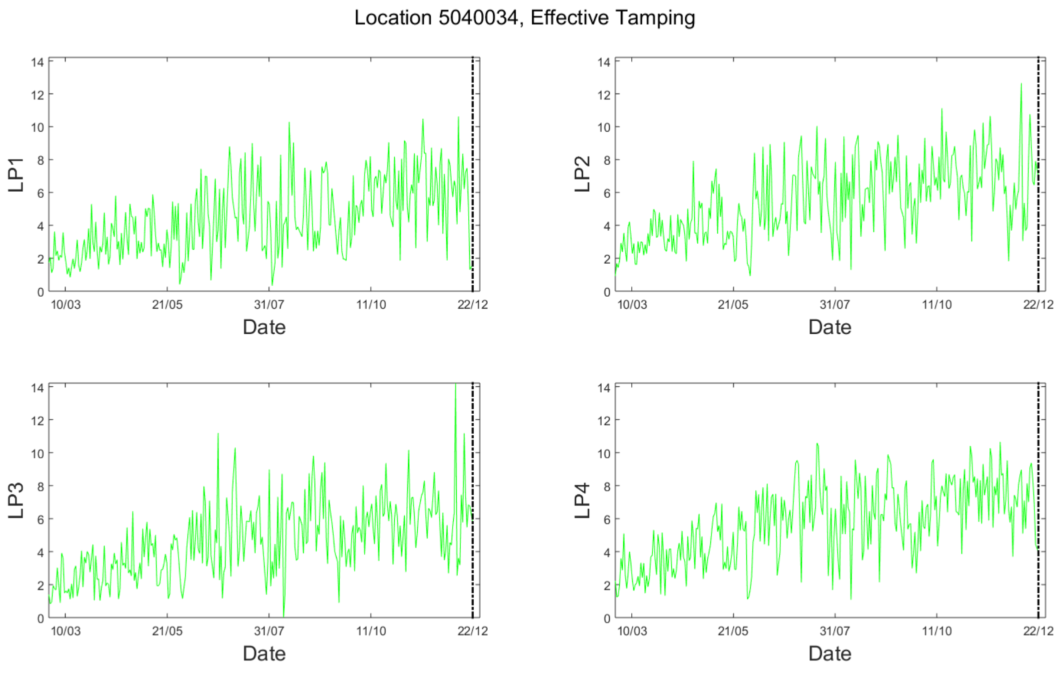


Figure 7.3: Example of the 4-dimensional time series (LP$_1$, LP$_2$, LP$_3$, LP$_4$) extracted from IRVs responses in the training database

Figure 7.3 shows a typical example of the 4-dimensional time series, LP$_1$, LP$_2$, LP$_3$, LP$_4$ extracted from the IRVs responses stored in the database. The black dotted line at the end of each time series indicates tamping date. The time series has a continuous rising trend before tamping.

7.4.2 Predicting Tamping Effectiveness

In this study, the k nearest neighbour classifier coupled with DTW is used to classify (or predict) the tamping effectiveness for a track location. This is achieved by searching for the k most similar time series of the targeted location from the time series database and returns the dominating tamping effectiveness as the tamping effectiveness of the targeted location.

Algorithm 18: PREDICTEFFECTIVENESS(S, $\mathcal{T}$)

```
Input: S: Query time series
Input: $\mathcal{T}$: Training dataset
Result: $\varepsilon$: Tamping effectiveness
/* all elements in $knn$ is initialised with no NN and $\infty$ distance to S */
1 Initialise $knn$ queue with $(-, \infty)$
2 foreach T $\in \mathcal{T}$ do
3     $d = \text{DTW}^D(\texttt{S}, \texttt{T})$
4     if $d < knn.\texttt{top}$ then
5         $knn.\texttt{remove}$                    // remove the furthest neighbour
6         $knn.\texttt{add}(T, d)$
7     end
8 end
9 return $\varepsilon = \texttt{mode}(knn.\texttt{class})$
```

The algorithm used in this work to predict a location with a tamping effectiveness is shown in Algorithm 18. The algorithm is initialized by creating two lists for storing the k-NN distances and class (effectiveness). The $knn.\texttt{distance}$ stores the distances while $knn.\texttt{class}$ store the class of the k-NN found. The algorithm proceeds by comparing the target time series with all instances in the training database and updates the two result lists. Finally, the dominating effectiveness in the k closest neighbours is the effectiveness of the target.

The k value in the k-NN classifier is very important in determining the performance of the classifier. Thus, it will need to be trained for the optimal k value. This is commonly done by dividing the whole database into training and testing sets. This technique is known as cross validation and will be described in the next section. Different k values are selected to train for the optimal k that gives the best classification performance. Then, the test set is used to verify and validate the performance of all the k values.

In this study, k is varied from 1 to 13, using only the odd numbers. It is important to note that k-NN is sensitive to the similarity measure used. Only DTW distance is considered as the responses from IRVs were too complicated for the Euclidean distance to handle. Hence, it is expected that DTW will outperform ED.

7.4.3 Training and Testing: Cross Validation

There are many techniques to divide the database into training and testing sets. Here, stratified K-fold cross validation is used to train and validate the best k value for the k-NN classifier. K-fold cross validation reduces the bias of training and testing data and has been used extensively as a method to estimate the generalization error based on 're-sampling'. The cross-validation technique estimates the learning ability of a classifier from a training dataset to classify future unseen data in the testing phase.

Generally, stratified K-fold cross validation divides the database into K subsets of equal size where each subset contains an equal distribution of classes. Then, the classifier is trained and tested K times, where each time, one subset will be used as the testing set while the rest $K - 1$ subsets will be used to train the classifier. A stratified 10-fold cross validation was employed for all the experiments in this study.

7.4.4 Performance Evaluation

The classification is evaluated using only two classes, positive (effective) and negative (ineffective). Then the four subsets of binary classification results as shown in Table 7.2 are reported.

Actual \ Predict	Effective tamping	Ineffective tamping
Effective tamping	True Positive	False Negative
Ineffective tamping	False Positive	True Negative

Table 7.2: Evaluation concept of classification results (effective tamping as positive class)

1. True positive (TP) denotes the correct classification of positive class.
2. True negative (TN) denotes the correct classification of negative class.
3. False positive (FP) denotes the incorrect classification of negative class into positive class.
4. False negative (FN) denotes the incorrect classification of positive class into negative class.

Then, the total TP, FP, TN, and FN for all the 10-folds is used to compute 4 performance metrics: accuracy (Equation 7.4), precision (Equation 7.5), sensitivity (Equation 7.6) and F_1 score (Equation 7.7) as defined as follow:

$$
\begin{aligned}
\text{accuracy} &= \frac{TP+TN}{TP+FP+TN+FN} && (7.4)\\
\text{precision} &= \frac{TP}{TP+FP} && (7.5)\\
\text{sensitivity} &= \frac{TP}{TP+FN} && (7.6)\\
F_1 \text{ score} &= 2\frac{\text{precision} \cdot \text{sensitivity}}{\text{precision} + \text{sensitivity}} && (7.7)
\end{aligned}
$$

Accuracy measures the systematic error of the classifier and the ratio of correct predictions to the total number of cases examined. Accuracy is not however sufficient to describe the performance of the classifier. For this it is better to consider other metrics such as precision, sensitivity and F_1. Precision (positive predictive value), measures the random error of the classifiers. Sensitivity (recall) is the true positive rate of the classifier, it measures the proportion of positive classes that are correctly being predicted. A similar metric, Specificity, is the true negative rate of the classifier. The F_1 score is the weighted average of precision and sensitivity. It is often used in binary classification to measure the classifierś accuracy. It ranges from 0 to 1, with 1 being the best.

The precision, sensitivity and F_1 score for both effective and ineffective tamping are computed in the experiments. This is done by treating one class as positive and the other as negative and vice versa. It is also worth noting that sensitivity for ineffective tamping is the specificity for effective tamping.

7.5 Results and Discussion

7.5.1 Classifiers Comparison

The NN-DTW classifier has shown to be very effective and competitive in many applications, as discussed above. This experiment shows that it is also the case for this task, even for multivariate time series. In this experiment, full length time series are used.

k-NN-DTW with different k values are compared with two other time series classification algorithms, the Naïve Bayes, and Classification Tree algorithm. These classifiers are different from the instance based k-NN-DTW that uses a similarity metric to classify a target time series. They use the 'features' of the time series as the input to the classifier to train a model that describes the training data. In this case, 6 statistical features, mean, standard deviation, skewness, kurtosis, maximum and minimum value of the time series are chosen.

The classifiers that will be compared in this experiment are described as follows:

- Naïve Bayes classifier is a popular probabilistic classification algorithm based on Bayes' theorem. It assumes that all the features are independent of one another and assigns probabilities of an instance to each of the class. The target is assigned to the class with the highest posterior.
- Classification Tree is also a common classification algorithm. It builds a graph model based on the features of the training database and maps the features of the target to the targeted class, which is represented in the leaves. Matlab classification tree function `fitctree` is used in this work to build the tree and classify the target.
- k-NN-DTW as described in Section 7.3.3 is used with $k = [1, 3, 5, 7, 9, 11, 13]$

The evaluation metrics described in the Section 7.4.4 are used to identify the best classification algorithm and optimal k value. The TP, FP, TN, FN are reported for all the classifiers and for all the 10-folds.

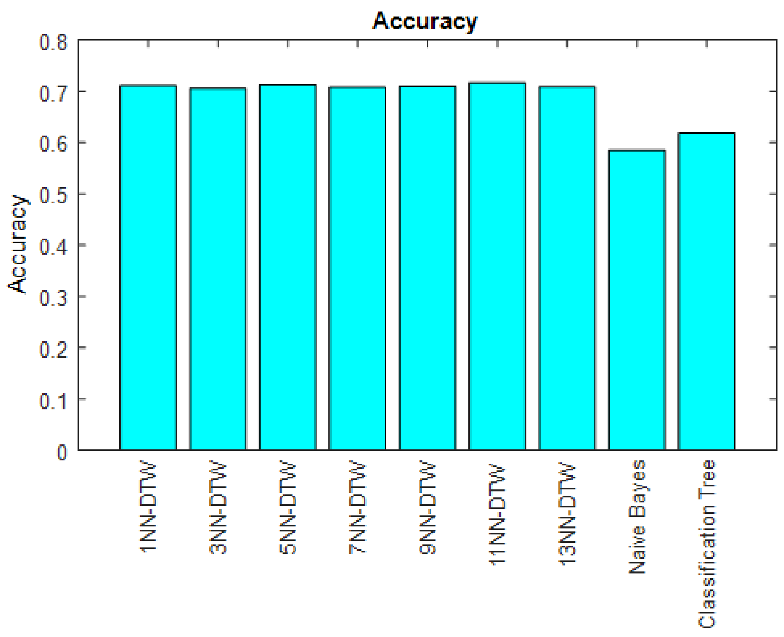
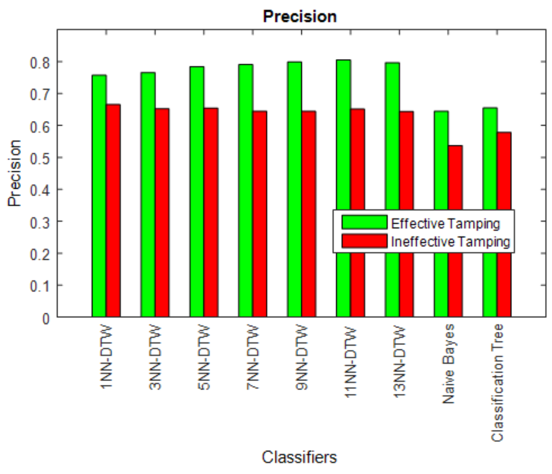


Figure 7.4: Accuracy of the different classifiers

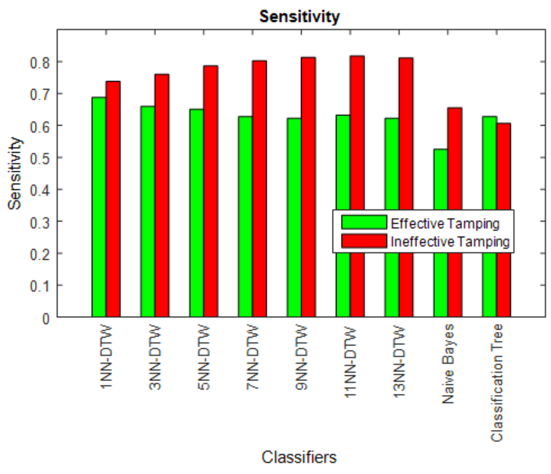
Figure 7.4 shows the accuracy of the different classifiers. This shows that k-NN achieves higher accuracy than the other two classifiers with an average accuracy of 71% compared to 58% and 62% for Naïve Bayes and Classification Tree. The highest being 11-NN with accuracy of 72%.

Figure 7.5a and 7.5b show the precision and sensitivity of predicting effective and ineffective tamping respectively. The two plots show that k-NN has better performance than the Naïve Bayes and Classification Tree classifier. In terms of precision, k-NN has an average precision of 78% in predicting effective tamping and 65% in predicting ineffective tamping. This is higher than the other two, where the highest precision among the two for both effective and ineffective tamping is only 65%. k-NN is also more sensitive than the other two, with an average sensitivity of 79% for ineffective tamping and 64% (higher for smaller k) for effective tamping. The highest among the other two for both effectiveness is only 65%.

Lastly, Figure 7.6 shows the F_1 score for all the classifiers. The plot shows that k-NN has a higher F_1 score for effective tamping with an average of 70% compared to 57% and 64% for the other two. It also has higher average F_1 score of 71% in predicting in-effective tamping compared to the other two at 59%. This means that it is better in predicting both effective and ineffective tamping than the other two.



(a)



(b)

Figure 7.5: (a) Precision and (b) Sensitivity of the different classifiers

A higher accuracy means that k-NN can predict tamping effectiveness well for all the locations on track. However, as mentioned in previous section, this is not sufficient. High precision and sensitivity in both effectiveness means that k-NN can identify those locations on track where tamping that will and will not be effective. Higher F_1 score indicates that k-NN performs better in predicting tamping effectiveness than the other two classification algorithm.

The other two classifiers are not able to predict well for two reasons. Firstly, the time series ranges from 37 to 184 days which is not long enough to have sufficient information that can be captured as features. Secondly, even if there are enough features to be captured, they are static. However in reality, these features tend to change dynamically.

It is important to observe that in this case, k does not affect the performance of k-NN. Hence, the k value that gives the best accuracy is chosen as the optimal k for the k-NN classifier. In this case, $k = 11$ with accuracy of 71% is chosen and was used in the next experiment. It is also worth noting that, the performance of k-NN will improve with the size of the database. The bias for k-NN vanishes in the limit as the size of database increases. This should make k-NN a more suitable classifier for predicting tamping effectiveness with railway data.

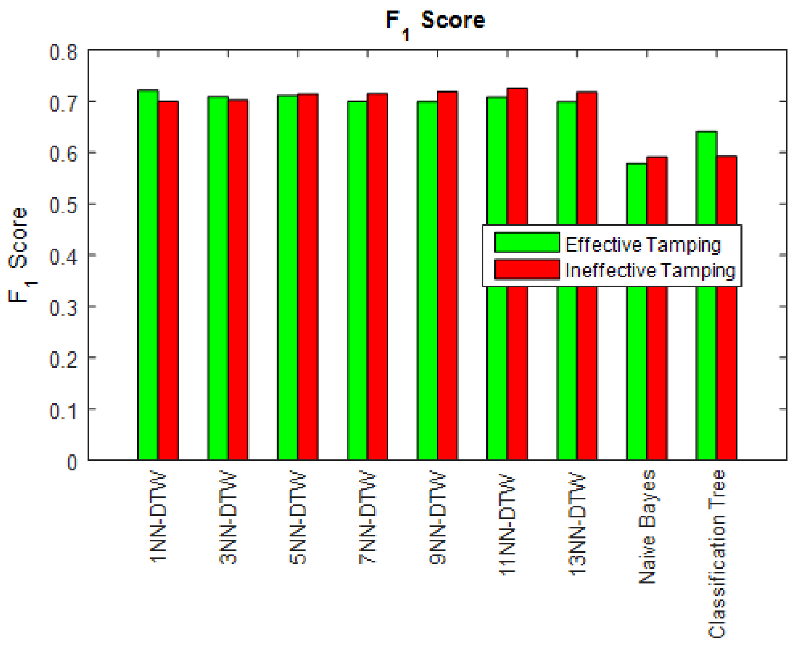


Figure 7.6: F_1 score of the different classifiers

7.5.2 Early Prediction of Tamping Effectiveness

Being able to predict the tamping effectiveness as early as possible is important in maintenance planning. The earlier we know about the likely tamping effectiveness; the better alternative maintenance activities can be planned. This section presents a study on the early prediction of tamping effectiveness using the 11-NN-DTW algorithm.

The study uses the same time series database that was used in previous experiment. The study was conducted as follow:

1. Truncate the existing time series by the number of days before tamping. Locations with time series that are shorter than the truncated length are not considered.
2. Run 11-NN-DTW to predict the tamping effectiveness of the targeted location.
3. Repeat with different number of days before tamping, ranging from 0 to 84 days (12 weeks). Note that, it is common to plan for tamping 12 weeks in advance.

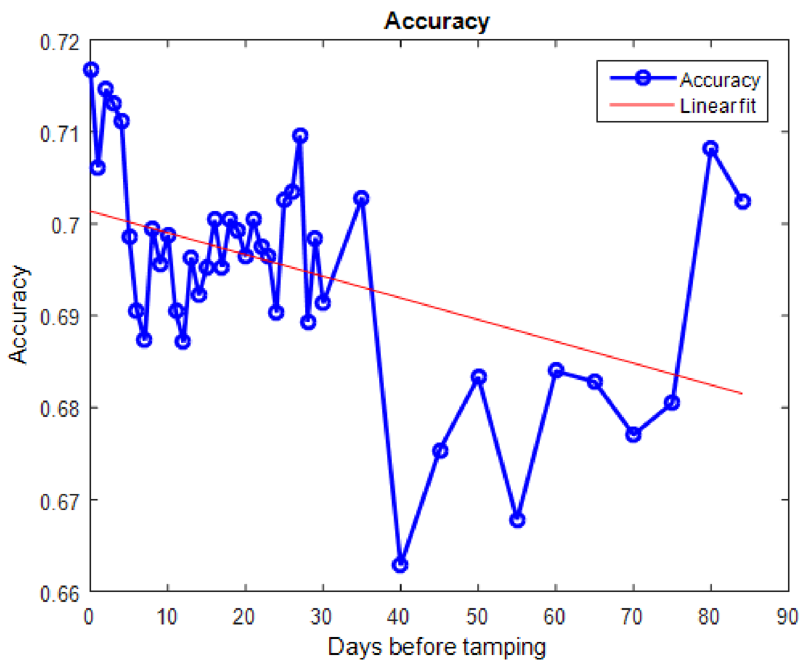


Figure 7.7: Accuracy for early prediction of tamping effectiveness at different days before tamping

Figure 7.7 shows the accuracy of predicting tamping effectiveness at different number of days before tamping. As expected, the accuracy decreases as the number of days before tamping increases. In other words, the time series are getting shorter, which means that there is less information and patterns that can be extracted from the time series itself, resulting in a reduction of accuracy. A simple linear regression suggests that the accuracy of predicting tamping effectiveness as early as 12 weeks before tamping is 68%.

7.6 Conclusion

This study presents an approach to predict tamping effectiveness using the k-nearest neighbour classifier. The prediction of tamping effectiveness should improve the efficiency of railway maintenance. The prediction approach has been assessed and validated using 10-fold cross validation on real-world rail data. This approach was found to achieve an accuracy of around 70% when predicting tamping effectiveness one day before tamping and 68% for 12 weeks before tamping. T his approach should help many railway corporations minimize maintenance costs, reduce unplanned downtime, and avoid cost of failure recovery by knowing the likely tamping effectiveness ahead of time. Future work includes optimizing the approach with more data, using a weighted k-NN approach as well as a stacked classifier to improve the confidence in the prediction.

7.7 Acknowledgement

We would like to thank Mr Cameron Thompson and Mr Joshua White from the Institute of Railway Technology (IRT), Monash University for their assistance in preparing the data.

Chapter 8

Concluding Remarks

8.1 Conclusion

As time series data is growing at an exponential rate, mining large time series databases is not a trivial task. Various algorithms had been studied and proposed in this thesis to scale up time series classification to the range of millions and billions. In conclusion, this thesis made the following contributions to the field of scalable time series classification.

A novel and efficient indexing algorithm, *Time Series Indexing* (TSI) is proposed in Chapter 3 to index time series data in DTW-induced space. The TSI algorithm indexes time series data using a hierarchical K-means tree structure, which was not possible before due to K-means clustering being ill-defined for DTW-induced space. To overcome this limitation, TSI leverage off a recent work in time series averaging – DTW Barycenter Averaging [47] to perform K-means clustering in DTW-induced space. Experimental results show that using TSI out of the box (using 3 clusters per tree level) is able to reduce the classification time of NN-DTW by 1,000 times on the one million SITS dataset [1]. The actual number of clusters is domain specific, which can be learned by testing different values of K and selecting the K value that gives the minimum time per query. More importantly, if given only 50% of the full classification time, TSI is significantly more accurate than NN-DTW on the 84 benchmark datasets and the SITS dataset tested.

Chapter 4 in this thesis proposed the *Fast Warping Window Search* algorithm (FASTWWS), a novel, exact and efficient learning algorithm to learn the best warping window for the NN-DTW classifier. FASTWWS improves upon existing state of the art learning algorithms by exploiting the relationship between DTW and its warping window. This work introduces new bounds that can be used across multiple windows, which has the effect of pruning many of the redundant distance computations. FASTWWS is always two to three orders of magnitude faster than existing learning algorithms. This work is important because it provides a basis to speed up the learning process of other NN classifiers in EE, which is an important module for the most accurate TSC algorithm, HIVE-COTE.

The *Fast Ensembles of Elastic Distances* (FASTEE) is proposed in Chapter 5 by extending FASTWWS to other distance measures. FASTEE exploits the relationship between each of the distance measures and their parameters. This work also introduces new lower bounds for distances for which no previous bounds have been derived, which is critical to speeding up the NN algorithm. Specifically lower bounds for WDTW, MSM, and TWED are introduced. An exact and approximate version of FASTEE are proposed. The exact FASTEE learns the same parameters as EE and is 10 times faster than EE. The approximate FASTEE approximates the leave-one-out cross validation process in EE and might not learn the same parameters as EE. However, it is 40 times faster than EE without significantly compromising on the classification accuracy.

The work in Chapter 6 proposes the *Enhanced Lower Bound* (LB_ENHANCED) to lower bound DTW. LB_ENHANCED exploits the tight boundary conditions in DTW to achieve a tighter lower bound, while having $O(L)$ complexity. The speed-tightness trade-off of LB_ENHANCED results in faster lower bound search for NN-DTW than any of the previous established bounds at all window sizes. The experimental results show that choosing a small tightness parameter $V = 5$ is sufficient to effectively speed up NN-DTW. It is also possible to learn the best V for a dataset, which will be future work. Having a tighter DTW lower bound will improve the effectiveness of nearest neighbour retrieval tasks under DTW. This includes learning the best warping window, which usually requires the enumeration of 100 windows in the range of 0 to L.

Finally, with two papers published in top railway conferences and receiving good feedback from the railway industry, this thesis also successfully shows that TSC can be applied to predict track maintenance activity, particularly the tamping maintenance. Tamping effectiveness is done by classifying time series taken from the Instrumented Revenue Vehicles, where each of them is labelled with their tamping effectiveness. The k-NN-DTW algorithm is used as the classification algorithm. The algorithm is able to predict tamping effectiveness 12 days before a tamping maintenance. This work is significant because it can minimise maintenance cost by allowing railway engineers to plan and schedule the suitable maintenance activity ahead of time.

8.2 Limitations

Despite the contributions and successes in the work presented in this thesis, there are a few limitations that should be addressed.

1. Building the indexing structure for TSI is very time consuming and not feasible especially for large datasets.
2. Although the FASTWWS algorithm has successfully reduced the training time for NN-DTW by two orders of magnitude, it is still impractical for many real-world applications. For instance, learning the best warping window in 6 hours (see Section 4.4) is still not very practical. The same limitation applies to FASTEE as well.
3. The experimental results show that NN-DTW when paired with LB_ENHANCED is the fastest compared to other DTW lower bounds. However, the speed up gained is still not fast enough for large datasets.
4. A simple study (not presented in this thesis) found that the length of the time series (maintenance history) is a major factor in predicting tamping effectiveness, i.e. shorter time series are likely to have an ineffective tamp.

8.3 Future Work

This section outlines the potential future directions to address the limitations mentioned. This section will also discuss about the some of the directions for time series classification.

As building TSI structure is time consuming, methods to reduce the time such as bulk loading mechanism in iSAX2.0 [35] can be considered. The possibility of extending TSI and state-of-the-art TSC algorithms to multivariate time series will be beneficial as many real-life applications are multivariate in nature. TSI can also be modularised so that it can be used with other clustering algorithms that might be better depending on the application. Since the training of FASTWWS and FASTEE requires NN searches, indexing techniques such as TSI can be applied to speed up the process. This means that TSI will need to be extended to other elastic distances. Furthermore, tighter lower bounds can be investigated for other elastic distances to speed up FASTEE. A possible direction is to extend LB_ENHANCED to other elastic distances, since most elastic distances share similar properties. It might also be possible to further speed up LB_ENHANCED using LB_IMPROVED instead of LB_KEOGH.

The exploration of alternatives, possibly HIVE-COTE can also be used to improve the prediction of tamping effectiveness, while taking into consideration of the length of the time series. A recent work on MATRIX PROFILE [132] can also be considered to improve the prediction.

Overall, time series classification and time series analysis have gained lots of interest in the recent years. Hundreds of TSC algorithms have been proposed in the past two decades. Applications like railway maintenance are starting to look into applying TSC to solve their problems and improve their systems. With the advancement in deep learning models, traditional classification algorithms are being outperformed or at least having similar performance [68]. Deep learning models will be a very interesting direction for TSC as it has great potentials for superior performance and the ability for transfer learning. Last but not least, new scalable algorithms such as the *Proximity Forest* [67] has the potential to outperform HIVE-COTE without the high complexity of HIVE-COTE.

Appendix A

Fail-Safe Experiment for FastWWS

Table A.1 shows the warping window learnt by all the methods on some of the UCR benchmark datasets [6] using exhaustive search (searching all possible warping windows). Please refer to `http://bit.ly/SDM18` for the full detailed results. Note that the results reported here are the actual warping window and not a percentage of the L (commonly done in the literature [6, 92, 133]. As expected, all the methods learnt the same warping window.

Figure A.1 shows the classification accuracy on the UCR Benchmark datasets [6] using the best warping window found for each individual method. Since all the methods are exact and that we are performing an exhaustive search (i.e. finding all possible warping windows $w = \{0, 1, 2, ..., L\}$), the best warping window found for each method is the same. Hence, the classification accuracy is the same. The only difference is the time which can be referred to Figure 4.4 in Chapter 4.

	Best warping window learnt by the following methods				
Datasets	LB_KEOGH	UCR_SUITE	LB_KEOGH–PRUNEDDTW	UCR_SUITE–PRUNEDDTW	**FASTWWS**
50words	24	24	24	24	24
Adiac	6	6	6	6	6
ArrowHead	0	0	0	0	0
Beef	0	0	0	0	0
BeetleFly	36	36	36	36	36
BirdChicken	33	33	33	33	33
CBF	14	14	14	14	14
Car	9	9	9	9	9
ChlorineConcentration	0	0	0	0	0
CinC_ECG_torso	10	10	10	10	10
Coffee	0	0	0	0	0
Computers	74	74	74	74	74
Cricket_X	31	31	31	31	31
Cricket_Y	47	47	47	47	47
Cricket_Z	15	15	15	15	15
DiatomSizeReduction	0	0	0	0	0
DistalPhalanxOutlineAgeGroup	1	1	1	1	1
DistalPhalanxOutlineCorrect	2	2	2	2	2
DistalPhalanxTW	0	0	0	0	0
ECG200	0	0	0	0	0
ECG5000	1	1	1	1	1
ECGFiveDays	0	0	0	0	0
Earthquakes	17	17	17	17	17
ElectricDevices	13	13	13	13	13
FISH	19	19	19	19	19
FaceAll	4	4	4	4	4
FaceFour	6	6	6	6	6
FacesUCR	16	16	16	16	16
FordA	2	2	2	2	2
FordB	6	6	6	6	6
Gun_Point	0	0	0	0	0
Ham	1	1	1	1	1
HandOutlines	28	28	28	28	28
Haptics	21	21	21	21	21
Herring	18	18	18	18	18
InlineSkate	41	41	41	41	41
InsectWingbeatSound	2	2	2	2	2
ItalyPowerDemand	0	0	0	0	0
LargeKitchenAppliances	676	676	676	676	676
Lighting2	35	35	35	35	35
Lighting7	17	17	17	17	17
MALLAT	0	0	0	0	0
Meat	0	0	0	0	0
MedicalImages	20	20	20	20	20
MiddlePhalanxOutlineAgeGroup	4	4	4	4	4
MiddlePhalanxOutlineCorrect	1	1	1	1	1
MiddlePhalanxTW	2	2	2	2	2
MoteStrain	1	1	1	1	1
NonInvasiveFatalECG_Thorax1	8	8	8	8	8
NonInvasiveFatalECG_Thorax2	0	0	0	0	0
OSULeaf	30	30	30	30	30
OliveOil	0	0	0	0	0
PhalangesOutlinesCorrect	0	0	0	0	0
Phoneme	143	143	143	143	143
Plane	8	8	8	8	8
ProximalPhalanxOutlineAgeGroup	0	0	0	0	0
ProximalPhalanxOutlineCorrect	1	1	1	1	1
ProximalPhalanxTW	5	5	5	5	5
RefrigerationDevices	54	54	54	54	54
ScreenType	122	122	122	122	122
ShapeletSim	7	7	7	7	7
ShapesAll	27	27	27	27	27
SmallKitchenAppliances	117	117	117	117	117
SonyAIBORobotSurface	0	0	0	0	0
SonyAIBORobotSurfaceII	0	0	0	0	0
StarLightCurves	65	65	65	65	65
Strawberry	1	1	1	1	1
SwedishLeaf	5	5	5	5	5

Table A.1: Learnt warping window from all the methods on the majority of the Benchmark datasets [6]

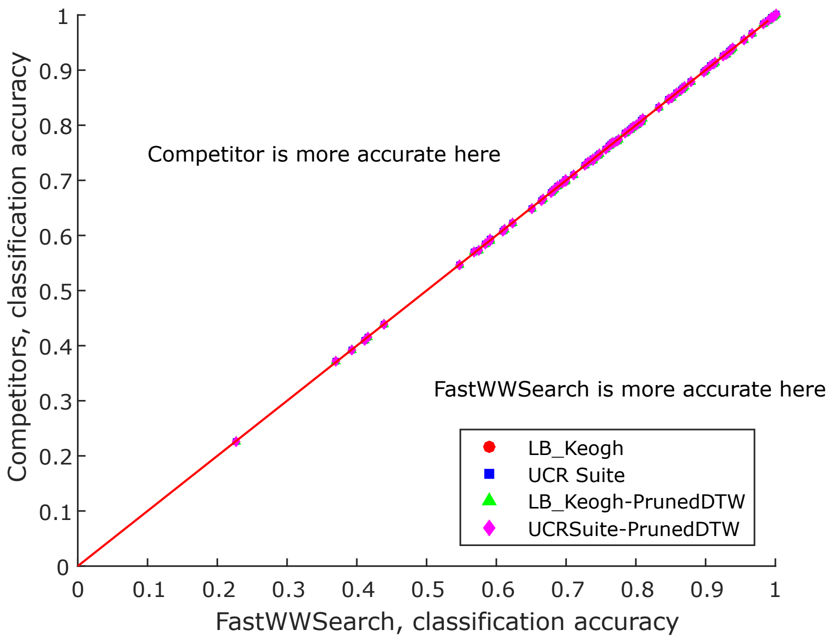


Figure A.1: Classification accuracy on the UCR Benchmark datasets [6] using the best warping window found for each method

Appendix B

Comparison of all DTW lower bounds

We used all the $N{=}85$ UCR benchmark datasets [6] and the given train/test splits to compare the performance of all $k{=}5$ lower bounds. For each dataset, lower bound and window size, we record the total classification time. Classification time is dependent on the ordering of the training set, the results were averaged again over 10 runs, shuffling the training data each time. Then we rank each of the lower bound for each of the dataset and compute the average rank of the lower bounds across all datasets. For a given window, W, we let r_i^j be the rank of the j-th lower bound on the i-th dataset. Then the average rank of the lower bounds is computed as $R_j = \frac{1}{N}\sum_{i=1}^{N} r_i^j$. The average rank gives a general comparison between the lower bounds, with rank 1 being the best and rank 5 being the worst. In case of ties, we assign the average rank to both bounds. Then the average rank for each bound is computed across all the datasets.

We also performed a Friedman test [119] to test the significance of our results. The Friedman test compares the average rank of the lower bounds and evaluates if the lower bounds are not performing equally using the Friedman statistic [119] described in Equation B.1.

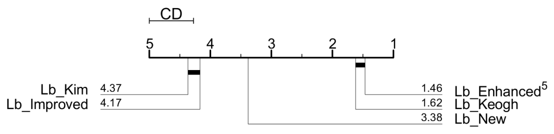
$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{B.1}$$

Using 5 lower bounds, the Friedman statistic is distributed according to χ_F^2 with 4 degrees of freedom. To reject the null hypothesis (all lower bounds perform equally) with α=0.05, the Friedman statistic has to be larger than the critical value of 9.49. If we reject the null hypothesis, then we proceed with the post-hoc test using the two-tailed Bonferroni-Dunn test [119]. Here, we have to compute the critical difference between a pair of lower bounds using Equation B.2.

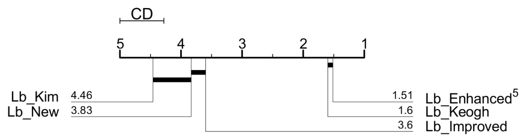
$$\text{CD} = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{B.2}$$

Comparing 5 lower bounds across 85 datasets and with α=0.05, q_α=2.728 [119]. Then to be statistically significant, the difference in rankings between two lower bounds has to be greater than the critical difference, CD=0.6616. In other words, if $R_1 > R_2$ and $R_1 - R_2 >$ CD, we say that LB$_1$ performs significantly better than LB$_2$.

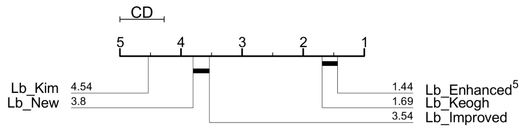
Figure B.1 shows the average ranking of all lower bounds in terms of NN-DTW classification time for $w = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Initially at smaller window sizes, there are no significant differences between LB_KEOGH and LB_ENHANCED^5. As window increases (at $w = 6$), LB_ENHANCED^5 starts to be significantly faster than LB_KEOGH. For larger window sizes, the main competitor for LB_ENHANCED^5 is LB_IMPROVED. Figure B.2 shows that for W=$\{0.1{\cdot}L, \ldots, 0.4{\cdot}L\}$, there are no differences between LB_IMPROVED and LB_ENHANCED^5. However, using NN-DTW with LB_ENHANCED^5 after $w = 0.5 \cdot L$ proves to be significantly faster than with other lower bounds.



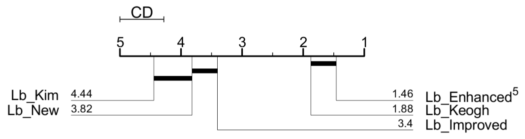
(a) $W\!=\!1$



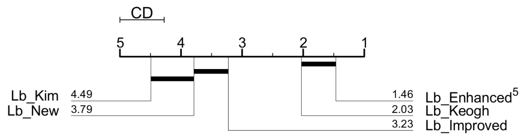
(b) $W\!=\!2$



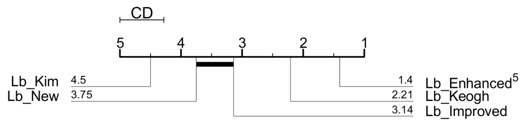
(c) $W\!=\!3$



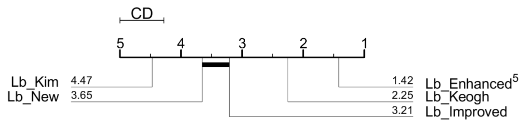
(d) $W\!=\!4$



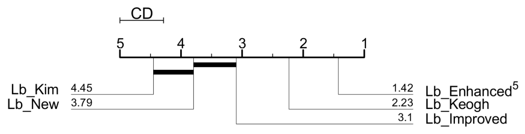
(e) $W\!=\!5$



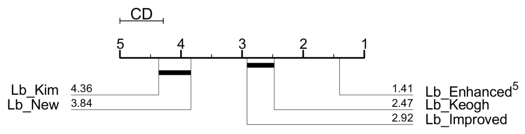
(f) $W\!=\!6$



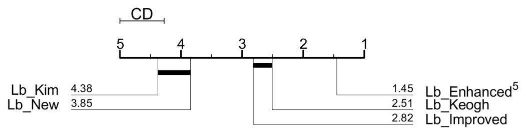
(g) $W\!=\!7$



(h) $W\!=\!8$

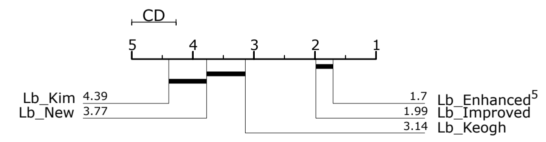


(i) $W\!=\!9$

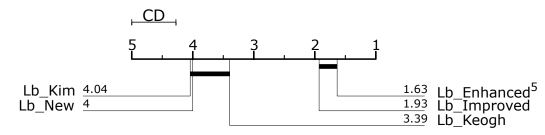


(j) $W\!=\!10$

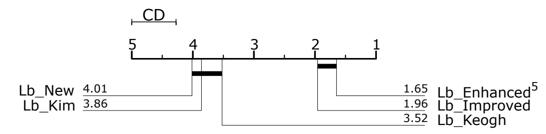
Figure B.1: Ranking of all lower bounds in terms of NN-DTW classification time for $w = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$



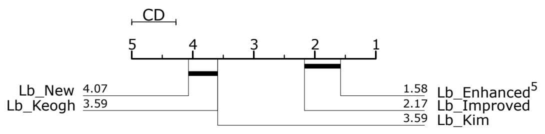
(a) $W=0.1 \cdot L$



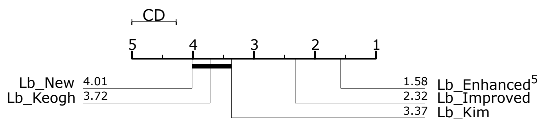
(b) $W=0.2 \cdot L$



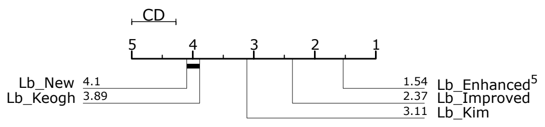
(c) $W=0.3 \cdot L$



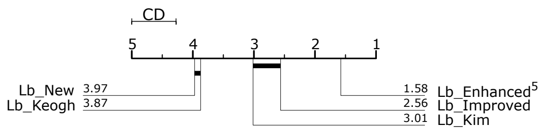
(d) $W=0.4 \cdot L$



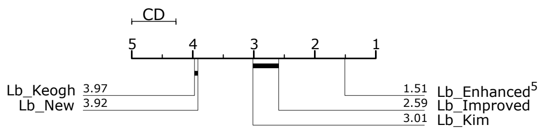
(e) $W=0.5 \cdot L$



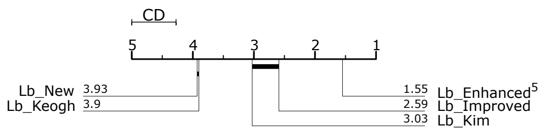
(f) $W=0.6 \cdot L$



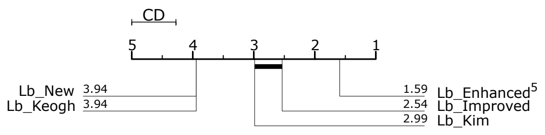
(g) $W=0.7 \cdot L$



(h) $W=0.8 \cdot L$



(i) $W=0.9 \cdot L$



(j) $W=L$

Figure B.2: Ranking of all lower bounds in terms of NN-DTW classification time for $w = \{0.1 \cdot L, 0.2 \cdot L, 0.3 \cdot L, 0.4 \cdot L, 0.5 \cdot L, 0.6 \cdot L, 0.7 \cdot L, 0.8 \cdot L, 0.9 \cdot L, L\}$

Bibliography

[1] C. W. Tan, G. I. Webb, and F. Petitjean, “Indexing and classifying gigabytes of time series under time warping,” in *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 282–290, SIAM, 2017.

[2] C. W. Tan, G. I. Webb, F. Petitjean, and P. Reichl, “Tamping effectiveness prediction using supervised machine learning techniques,” in *Proceedings of the First International Conference on Rail Transportation (ICRT)*, 2017.

[3] C. W. Tan, M. Herrmann, G. Forestier, G. I. Webb, and F. Petitjean, “Efficient search of the best warping window for dynamic time warping,” in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 225–233, SIAM, 2018.

[4] C. W. Tan, P. Reichl, G. I. Webb, and F. Petitjean, “Machine learning approaches for tamping effectiveness predictions,” in *Proceedings of the 2017 International Heavy Haul Association Conference (IHHA2017)*, IHHA, 2017.

[5] Forbes, “Really big data at walmart: Real-time insights from their 40+ petabyte data cloud.”

[6] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, “The UCR Time Series Classification Archive,” 7 2015. www.cs.ucr.edu/~eamonn/time_series_data/.

[7] E. S. Agency, “Sentinel 2 overview,” 2016. https://sentinel.esa.int/web/sentinel/missions/sentinel-2/overview.

[8] J. Inglada, A. Vincent, M. Arias, and C. Marais-Sicre, “Improved early crop type identification by joint use of high temporal resolution sar and optical image time series,” *Remote Sensing*, vol. 8, no. 5, p. 362, 2016.

[9] J. Inglada, M. Arias, B. Tardy, O. Hagolle, S. Valero, D. Morin, G. Dedieu, G. Sepulcre, S. Bontemps, P. Defourny, and B. Koetz, “Assessment of an operational system for crop type map production using high temporal and spatial resolution satellite optical imagery,” *Remote Sensing*, vol. 7, no. 9, pp. 12356–12379, 2015.

[10] F. Petitjean, J. Inglada, and P. Gançarski, “Satellite image time series analysis under time warping,” *IEEE transactions on geoscience and remote sensing*, vol. 50, no. 8, pp. 3081–3095, 2012.

[11] M. Darby, E. Alvarez, J. McLeod, G. Tew, and G. Crew, “The development of an instrumented wagon for continuously monitoring track condition,” in *AusRAIL PLUS 2003, 17-19 November 2003, Sydney, NSW, Australia*, 2003.

[12] M. Darby, E. Alvarez, J. McLeod, G. Tew, G. Crew, *et al.*, “Track condition monitoring: the next generation,” in *Proceedings of 9th International Heavy Haul Association Conference*, vol. 1, pp. 1–1, 2005.

[13] G. Hardie, G. Tew, G. Crew, S. Oswald, and M. Courtney, “Track condition assessment and monitoring in heavy haul railroads,” in *2011 IHHA Conference, Calgary*, 2011.

[14] J. Cowie, L. Taylor, B. Felsner, A. Stramare, and G. Hardie, "Use of instrumented revenue vehicle to manage 40 tonne axle load operation at fortescue metals group ltd," in *Proceedings of the 2015 11th International Heavy Haul Association (IHHA2015) Conference*, pp. 870–878, International Heavy Haul Association, 2015.

[15] J. Lines, S. Taylor, and A. Bagnall, "Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 1041–1046, Dec 2016.

[16] A. Bagnall and J. Lines, "An experimental evaluation of nearest neighbour time series classification. technical report# cmp-c14-01," *Department of Computing Sciences, University of East Anglia, Tech. Rep*, 2014.

[17] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 565–592, 2015.

[18] M. Audley and J. Andrews, "The effects of tamping on railway track geometry degradation," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of rail and rapid transit*, vol. 227, no. 4, pp. 376–391, 2013.

[19] I. Arasteh Khouy, *Cost-effective maintenance of railway track geometry: A shift from safety limits to maintenance limits*. 2013.

[20] C. Esveld, *Modern railway track*, vol. 385. MRT-productions Zaltbommel, The Netherlands, 2001.

[21] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with cote: the collective of transformation-based ensembles," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2522–2535, 2015.

[22] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[23] 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, no. 3, pp. 606–660, 2017.

[24] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 275–309, 2013.

[25] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, p. 12, 2012.

[26] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Dynamic time warping averaging of time series allows faster and more accurate classification," in *2014 IEEE International Conference on Data Mining*, pp. 470–479, IEEE, 2014.

[27] E. Keogh and C. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005.

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

[29] B.-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *Data Engineering, 1998. Proceedings., 14th International Conference on*, pp. 201–208, IEEE, 1998.

[30] Y. Shen, Y. Chen, E. Keogh, and H. Jin, "Accelerating time series searching with large uniform scaling," in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 234–242, SIAM, 2018.

[31] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern recognition*, vol. 42, no. 9, pp. 2169–2180, 2009.

[32] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 262–270, ACM, 2012.

[33] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[34] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of machine learning*, pp. 257–258, Springer, 2011.

[35] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh, "isax 2.0: Indexing and mining one billion time series," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 58–67, IEEE, 2010.

[36] J. Shieh and E. Keogh, "i sax: indexing and mining terabyte sized time series," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 623–631, ACM, 2008.

[37] F. K.-P. Chan, A. W.-c. Fu, and C. Yu, "Haar wavelets for efficient similarity search of time-series: with and without time warping," *IEEE Transactions on Knowledge & Data Engineering*, no. 3, pp. 686–705, 2003.

[38] B.-K. Yi and C. Faloutsos, "Fast time sequence indexing for arbitrary lp norms," in *VLDB*, vol. 385, p. 99, Citeseer, 2000.

[39] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, *Fast subsequence matching in time-series databases*, vol. 23. ACM, 1994.

[40] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 285–289, ACM, 2000.

[41] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Knowledge and information Systems*, vol. 3, no. 3, pp. 263–286, 2001.

[42] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Sigmod Record*, vol. 30, no. 2, pp. 151–162, 2001.

[43] F. Korn, H. V. Jagadish, and C. Faloutsos, "Efficiently supporting ad hoc queries in large datasets of time sequences," in *Acm Sigmod Record*, vol. 26, pp. 289–300, ACM, 1997.

[44] C. Ratanamahatana and E. Keogh, "Three myths about DTW data mining," in *SIAM SDM*, pp. 506–510, 2005.

[45] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *International Congress on Acoustics*, vol. 3, pp. 65–69, 1971.

[46] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

[47] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[48] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[49] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*. World scientific, 2014.

[50] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[51] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *ACM Sigkdd Explorations Newsletter*, vol. 12, no. 1, pp. 40–48, 2010.

[52] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, no. 27, pp. 79–80, 1995.

[53] P. Smyth, "Clustering sequences with hidden markov models," in *Advances in neural information processing systems*, pp. 648–654, 1997.

[54] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142–153, 2013.

[55] M. G. Baydogan, G. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 11, pp. 2796–2802, 2013.

[56] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," in *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 947–956, ACM, 2009.

[57] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proceedings of the 13th SIAM international conference on data mining*, pp. 668–676, SIAM, 2013.

[58] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, ACM, 2003.

[59] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 392–401, ACM, 2014.

[60] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, “Classification of time series by shapelet transformation,” *Data Mining and Knowledge Discovery*, vol. 28, no. 4, pp. 851–881, 2014.

[61] J. Lin, R. Khade, and Y. Li, “Rotation-invariant similarity in time series using bag-of-patterns representation,” *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.

[62] P. Senin and S. Malinchik, “Sax-vsm: Interpretable time series classification using sax and vector space model,” in *2013 IEEE 13th International Conference on Data Mining*, pp. 1175–1180, IEEE, 2013.

[63] P. Schäfer, “The boss is concerned with time series classification in the presence of noise,” *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.

[64] P. Schäfer and M. Högqvist, “Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets,” in *Proceedings of the 15th International Conference on Extending Database Technology*, pp. 516–527, ACM, 2012.

[65] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[66] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, “Rotation forest: A new classifier ensemble method,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006.

[67] B. Lucas, A. Shifaz, C. Pelletier, L. O’Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, “Proximity forest: An effective and scalable distance-based classifier for time series,” *arXiv preprint arXiv:1808.10594*, 2018.

[68] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” *arXiv preprint arXiv:1809.04356*, 2018.

[69] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[71] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[72] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[73] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[74] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, “Deep convolutional neural networks for lvcsr,” in *Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on*, pp. 8614–8618, IEEE, 2013.

[75] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[76] Z. Wang, W. Yan, and T. Oates, “Time series classification from scratch with deep neural networks: A strong baseline,” in *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 1578–1585, IEEE, 2017.

[77] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, “Exploiting multi-channels deep convolutional neural networks for multivariate time series classification,” *Frontiers of Computer Science*, vol. 10, no. 1, pp. 96–112, 2016.

[78] Z. Cui, W. Chen, and Y. Chen, “Multi-scale convolutional neural networks for time series classification,” *arXiv preprint arXiv:1603.06995*, 2016.

[79] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[80] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Transfer learning for time series classification,” *arXiv preprint arXiv:1811.01533*, 2018.

[81] 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.

[82] 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.

[83] C. A. Ratanamahatana and E. Keogh, “Making time-series classification more accurate using learned constraints,” in *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 11–22, SIAM, 2004.

[84] E. J. Keogh and M. J. Pazzani, “Derivative dynamic time warping,” in *Proceedings of the 2001 SIAM International Conference on Data Mining*, pp. 1–11, SIAM, 2001.

[85] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, “Weighted dynamic time warping for time series classification,” *Pattern Recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

[86] G. Das, D. Gunopulos, and H. Mannila, “Finding similar time series,” in *European Symposium on Principles of Data Mining and Knowledge Discovery*, pp. 88–100, Springer, 1997.

[87] M. Vlachos, G. Kollios, and D. Gunopulos, “Discovering similar multidimensional trajectories,” in *Data Engineering, 2002. Proceedings. 18th International Conference on*, pp. 673–684, IEEE, 2002.

[88] L. Chen and R. Ng, “On the marriage of lp-norms and edit distance,” in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pp. 792–803, VLDB Endowment, 2004.

[89] L. Chen, M. T. Özsu, and V. Oria, “Robust and fast similarity search for moving object trajectories,” in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 491–502, ACM, 2005.

[90] A. Stefan, V. Athitsos, and G. Das, “The move-split-merge metric for time series,” *IEEE transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1425–1438, 2013.

[91] P.-F. Marteau, “Time warp edit distance with stiffness adjustment for time series matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 306–318, 2009.

[92] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. Ratanamahatana, “Fast time series classification using numerosity reduction,” in *Proceedings of the 23rd international conference on Machine learning*, pp. 1033–1040, ACM, 2006.

[93] D. Silva and G. Batista, “Speeding up all-pairwise dynamic time warping matrix calculation,” in *SIAM SDM*, pp. 837–845, 2016.

[94] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[95] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration.,” *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[96] A. Andoni and P. Indyk, “E2lsh: Exact euclidean locality-sensitive hashing,” *Implementation available at http://web. mit. edu/andoni/www/LSH/index. html*, 2004.

[97] A. Andoni and P. Indyk, “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions,” in *Foundations of Computer Science, 2006. FOCS’06. 47th Annual IEEE Symposium on*, pp. 459–468, IEEE, 2006.

[98] B. Chiu, E. Keogh, and S. Lonardi, “Probabilistic discovery of time series motifs,” in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 493–498, ACM, 2003.

[99] L. Wei, E. Keogh, and X. Xi, “Saxually explicit images: Finding unusual shapes,” in *Sixth International Conference on Data Mining, ICDM’06*, pp. 711–720, IEEE, 2006.

[100] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, “Fast approximate nearest-neighbor search with k-nearest neighbor graph,” in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, p. 1312, 2011.

[101] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li, “Scalable k-nn graph construction for visual descriptors,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1106–1113, IEEE, 2012.

[102] R. Agrawal, C. Faloutsos, and A. Swami, “Efficient similarity search in sequence databases,” in *International conference on foundations of data organization and algorithms*, pp. 69–84, Springer, 1993.

[103] M. Flynn, J. Lines, and A. Bagnall, “c-rise: Contract random interval spectral ensemble for time series classification,” in *3nd ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2018.

[104] P. Schäfer, “Scalable time series classification,” *Data Mining and Knowledge Discovery*, pp. 1–26, 2015.

[105] H. Hamooni and A. Mueen, "Dual-domain hierarchical classification of phonetic time series," 2014. www.cs.unm.edu/~hamooni/papers/Dual_2014/index.html.

[106] H. Hamooni and A. Mueen, "Dual-domain hierarchical classification of phonetic time series," in *Data Mining (ICDM), 2014 IEEE International Conference on*, pp. 160–169, IEEE, 2014.

[107] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, "Indexing multi-dimensional time-series with support for multiple distance measures," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 216–225, ACM, 2003.

[108] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Proceedings of the seventh international congress on acoustics*, vol. 3, pp. 65–69, Budapest, Hungary, 1971.

[109] C. Sammut and G. Webb, eds., *Encyclopedia of Machine Learning*. Berlin: Springer, 2010.

[110] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas, "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures," *The VLDB journal*, vol. 18, no. 3, pp. 611–630, 2009.

[111] M. Muja, "Flann-fast library for approximate nearest neighbors," 2014. www.cs.ubc.ca/research/flann/.

[112] F. Petitjean, G. Forestier, G. Webb, A. Nicholson, Y. Chen, and E. Keogh, "Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm," *Knowledge and Information Systems*, vol. 47, no. 1, pp. 1–26, 2016.

[113] E. Keogh, "Welcome to the lb_keogh homepage!," 2001. www.cs.ucr.edu/~eamonn/LB_Keogh.htm.

[114] P. Kranen and T. Seidl, "Harnessing the strengths of anytime algorithms for constant data streams," *Data Mining and Knowledge Discovery*, vol. 19, no. 2, pp. 245–260, 2009.

[115] Y. Yang, G. Webb, K. Korb, and K.-M. Ting, "Classifying under computational resource constraints: Anytime classification using probabilistic estimators," *Machine Learning*, vol. 69, no. 1, pp. 35–53, 2007.

[116] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[117] J. L. Bentley, *Programming pearls*. ACM, 1986.

[118] "Additional material," 2017. http://bit.ly/SDM2017.

[119] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[120] H. Dau, D. Silva, F. Petitjean, A. Bagnall, and E. Keogh, "Judicious setting of DTW's warping window width allows more accurate classification of time series," in *IEEE Big Data Conference*, pp. 1–4, 2017.

[121] S. Srikanthan, A. Kumar, and R. Gupta, “Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery,” in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, pp. 394–398, IEEE, 2011.

[122] A. Shamdani, C. Thompson, F. Ahmed, and R. Penglase, “Analysis of track tamping effectiveness using continuously measured performance data,” in *Proceedings of the 2015 11th International Heavy Haul Association (IHHA2015) Conference*, pp. 1273–1278, International Heavy Haul Association, 2015.

[123] L. Baeza and H. Ouyang, “A railway track dynamics model based on modal substructuring and a cyclic boundary condition,” *Journal of Sound and Vibration*, vol. 330, no. 1, pp. 75–86, 2011.

[124] A. R. Andrade and P. F. Teixeira, “Hierarchical bayesian modelling of rail track geometry degradation,” *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of rail and rapid transit*, vol. 227, no. 4, pp. 364–375, 2013.

[125] H. Guler, “Prediction of railway track geometry deterioration using artificial neural networks: a case study for turkish state railways,” *Structure and Infrastructure Engineering*, vol. 10, no. 5, pp. 614–626, 2014.

[126] H. Guler, “Decision support system for railway track maintenance and renewal management,” *Journal of Computing in Civil Engineering*, vol. 27, no. 3, pp. 292–306, 2012.

[127] M. Shokoohi-Yekta, J. Wang, and E. Keogh, “On the non-trivial generalization of dynamic time warping to the multi-dimensional case,” in *Proceedings of the 2015 SIAM international conference on data mining*, pp. 289–297, SIAM, 2015.

[128] N. J. Nilsson, “Stuart russell and peter norvig, artificial intelligence: A modern approach,” *Artificial intelligence*, vol. 82, no. 1-2, pp. 369–380, 1996.

[129] H. Z. A. C. B. Michael and M. J. Malik, “Svmknn: Discriminative nearest neighbor classification for visual category recognition,” *Computer Science Division, EECS Department Univ. of California, Berkeley, CA*, vol. 94720, 2007.

[130] Z. Yong, L. Youwen, and X. Shixiong, “An improved knn text classification algorithm based on clustering,” *Journal of computers*, vol. 4, no. 3, pp. 230–237, 2009.

[131] L. Li, C. R. Weinberg, T. A. Darden, and L. G. Pedersen, “Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the ga/knn method,” *Bioinformatics*, vol. 17, no. 12, pp. 1131–1142, 2001.

[132] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, “Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets,” in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 1317–1322, IEEE, 2016.

[133] V. Niennattrakul and C. A. Ratanamahatana, “Learning dtw global constraint for time series classification,” *arXiv preprint arXiv:0903.0041*, 2009.