Time Series Analysis with Applications in Bioinformatics

時系列分析と生命情報学への応用

Coleman Yu

余　浩旻

Abstract

Time series data are ubiquitous across scientific disciplines, necessitating robust computational techniques for analysis and retrieval. This thesis addresses two critical challenges in the domain of time series data mining with applications in bioinformatics: the development of more expressive distance measures, with the usage of similarity search, and the novel application of time series classification to solve complex biological problems.

In the first part, we address the limitations of existing similarity measures. Similarity search is a core subroutine in tasks such as classification and motif finding. While Dynamic Time Warping (DTW) and Uniform Scaling (US) are prevailing measures for handling local distortions and global scaling, respectively, and some studies have demonstrated that combining both DTW and US is necessary to obtain meaningful results, the current approaches are limited to applying a single scaling factor to the entire sequence before applying DTW. We argue that since distinct phases of a process often evolve at different speeds, a single scaling factor is insufficient. We introduce the first distance measure that achieves invariance to multiple scaling factors. We also provide lower bounding techniques to facilitate efficient computation of the proposed distance measure. This method better reflects the similarity between time series with multiple phases and provides a clearer understanding of the data.

In the second part, we demonstrate the applied power of time series analysis within the field of bioinformatics. Bioinformatics operates at the intersection of Biology, Biotechnology, and Informatics. In this work, we formulate a specific

Biology problem, which is predicting Human Dicer Cleavage sites in microRNA biogenesis, into a machine learning framework. Due to the current limitations of Biotechnology, we are constrained to utilizing 1-D RNA sequence inputs rather than fully 2-D data; the latter is more expensive to obtain. We propose MTSC-Cleav, a method that encodes RNA sequences and the probabilities of base pairs in predicted secondary structures into time series data. By doing this, we frame the problem of predicting Human Dicer Cleavage sites into a Multivariate Time Series Classification (MTSC) problem. Unlike existing approaches that rely on opaque deep neural networks or complex feature engineering, our approach is simple and computationally efficient. Experiments demonstrate that MTSCCleav achieves comparable accuracy to state-of-the-art methods while delivering a 3.7X to 28.8X speedup. Furthermore, our perturbation experiments reveal that regions near the center of pre-miRNAs are essential for cleavage site prediction.

Collectively, this thesis advances the fields of time series data mining and bioinformatics by proposing a new, more expressive distance measure and demonstrating the use of time series analysis to address fundamental biological questions.

Acknowledgements

Publications

This thesis is based on the following papers.

- (Chapter 3) **Coleman Yu**, Raymond Chi-Wing Wong, Tatsuya Akutsu. MTSCCleav: a Multivariate Time Series Classification (MTSC)-based Method for Predicting Human Dicer Cleavage Sites. Submitted to *IEEE Access*, under review.
- (Chapter 4) **Coleman Yu**, Tatsuya Akutsu, Raymond Chi-Wing Wong. Scaling with Multiple Scaling Factors and Dynamic Time Warping in Time Series Searching. Submitted to *IEEE Access*, under review.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Preliminaries

Chapter 3

MTSCCleav: a Multivariate Time Series Classification (MTSC)-based Method for Predicting Human Dicer Cleavage Sites

MicroRNAs (miRNAs) are small non-coding RNAs (ncRNAs) that regulate gene expression at the post-transcriptional level, thereby playing essential roles in diverse biological processes. The biogenesis of miRNAs requires dicer to cleave at specific sites on the precursor miRNAs (pre-miRNAs). Several machine learning approaches have been proposed to predict whether an input sequence contains a cleavage site. However, they rely heavily on complex feature engineering or opaque deep neural networks. It results in a lack of generalizability and a long running time. There is a need for an alternative modeling paradigm that is accurate, fast, and simple.

We proposed a novel approach to frame the task as a multivariate time series classification problem. Nine encoding methods have been proposed to convert the sequence and the predicted secondary structure into a time series. We also leveraged the probabilities of the base pairs in the predicted secondary structure. Computational experiments demonstrate that our proposed method can achieve better or comparable results in terms of using a simpler, more intuitive model and less computational time. It achieves 3.7X to 28.8X speedup. Through perturbation experiments, we found that regions close to the center of pre-miRNAs are essential for predicting human dicer cleavage sites.

By transforming the RNA sequence and its secondary structure information into a time series and utilizing simple, state-of-the-art time series classifiers, we achieved comparable or even superior performance in a simpler and faster manner.

Code is available at: `https://github.com/colemanyu/time-series-classification-cleavage`.

3.1 Background

One of the most important theories in molecular biology is the central dogma. It depicts the flow of genetic information [62, 5]. Proteins are the functional units. The information stored in DNA is used to create them. Genes (segments) in DNA are used as templates for messenger RNAs (mRNAs) synthesis. An mRNA acts as a set of instructions to assemble a chain of amino acids, which form a linear polypeptide. To become biologically active, this chain is folded into a specific 3D structure, a proper configuration that enables it to perform its desired functions. This folded polypeptide is called a functional protein, or simply a protein. This entire process closely resembles how a computer program runs on a machine. The source code does not function by itself. First, it is translated into an assembly code (a lower-level, less human-readable form) and then into an executable file that can actually perform the intended tasks [12].

These mRNAs are called “coding RNAs” because they code for proteins. There are other genes in which the final product is the RNA molecule itself. They are called non-coding RNAs (ncRNAs). Two types of small ncRNAs are particularly important. They are microRNAs (miRNAs) and small interfering RNAs (siRNAs). Their discovery was recognized with the 2006 Nobel Prize in Physiology or Medicine[1], awarded for work completed only eight years prior [62].

In this study, we focus on miRNAs. An miRNA can regulate the expression of several proteins. Hence, understanding the biogenesis of miRNAs is of great value. It involves the processing of primary miRNAs (pri-miRNAs). RNAs are 3D molecules. However, it is hard to measure the 3D structure (tertiary structure) from the experiment and predict it from 1D sequence. We can understand their properties by analyzing their 1D sequence or 2D structure, known as secondary structure. RNA sequence is easily obtained through sequencing. The sequence and its predicted secondary structure of a pri-miRNA “hsa-let-7a-1” is shown in Figure 3.1.

[1]The Nobel Prize in Physiology or Medicine 2006 - NobelPrize.org: https://www.nobelprize.org/prizes/medicine/2006/summary/ (Accessed on: 2025-06-13).

[2]Its miRBase entry: https://mirbase.org/hairpin/MI0000060. (Accessed on: 2025-06-12).

[3]RNAfold web server: http://rna.tbi.univie.ac.at/cgi-bin/RNAWebSuite/RNAfold.cgi. (Accessed on: 2025-06-12). The figure is viewed in “forna”. This view option can be chosen on the website.

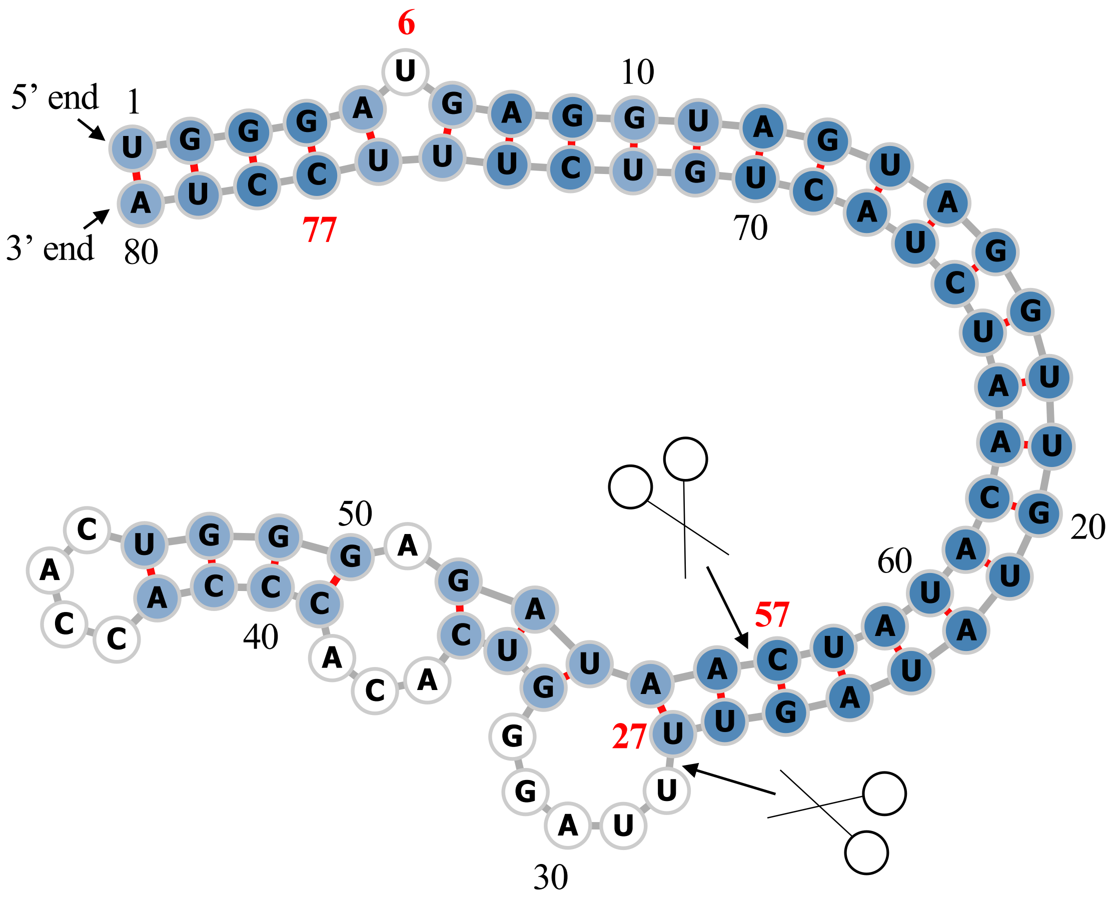


Figure 3.1: Predicted secondary structure of the sequence S of pri-miRNA "hsa-let-7a-1"[2]. Experimental evidence suggests that the two deviated mature miRNAs are $UGA \cdots GUU$ and $CUA \cdots UUC$. They are $S(6:27)$ and $S(57:77)$ (Both ends are inclusive.). The ends are highlighted in **bold**. Since $S(6:27)$ ($S(57:77)$) is near the 5' (3') end, we call it "5p (3p) mature miRNA". The two scissors indicate the two cleavage sites. The color intensity of the nodes reflects their base-pair probability in this predicted secondary structure. The deeper the color, the higher the probability. The unpaired nodes are uncolored. The raw figure is generated by RNAfold web server[3].

Recall that a pri-miRNA contains a hairpin loop, also called a stem loop. A microprocessor complex comprising Drosa and DCGR8 cleaves the pri-miRNA to form a precursor miRNA (pre-miRNA) inside the nucleus. The stem-loop is still preserved, but the two arms become shorter. After that, the pri-miRNA is transported by Exportin 5 from the nucleus to the cytoplasm. It is further cleaved by an enzyme called dicer [34]. Dicer cleaves the stem-loop from the two arms at the two cleavage sites, shown as the two scissors in Figure 3.1. The stem-loop is removed. It results in a short double-stranded miRNA molecule, known as an miRNA duplex, which consists of the 5p strand and the 3p strand[4]. These molecules may be subjected to additional trimming. The miRNA duplex is loaded into an RNA-induced silencing complex (RISC). RISC unwinds the duplex and tends to retain the strand with the less stable 5' end as the guide strand. The other strand is called the passenger strand. The retained strand guides the RISC to silence the target mRNA. Note that both strands can become the guide strand.

Dicer plays an important role in the biogenesis of miRNAs. It is reasonable to argue that the structure of the pre-miRNAs informs dicer about the cleavage process. It would be of great benefit to understand how dicer selects cleavage sites from the neighborhood information near the cleavage sites. Studies [25, 20, 40] revealed that the secondary structures are essential for cleavage site determination. Hence, to predict or classify whether a subsequence, extracted from the sequence of a pri-miRNA, contains a cleavage site, we can make use of both the sequence and secondary structure information. PHDcleav employed support vector machines (SVM), leveraging sequence and structure-based features for the classification [3].

[4]The 5p strand comes from the 5' arm while the 3p strand comes from the 3' arm. For the directionality, the 5p (3p) strand retains the original 5' (3') end of the pre-miRNA.

LBSizeCleav improved upon it by considering the loop and bulge lengths [8]. [38] proposed an ensemble learning approach, using a gradient boosting machine for better accuracy. [46] developed a deep learning model, namely DiCleave. This model used an autoencoder to learn the secondary structure embeddings of pre-miRNAs from all the species in the miRBase database and leveraged this information. All these methods begin with curated pre-miRNA sequences from the miRBase database. Their secondary structures are predicted. Patterns are extracted from the sequence and the secondary structure. They create the positive cleavage patterns by setting the cleavage sites at the middle of the patterns. The follow-up work of [46], which created the cleavage pattern by allowing cleavage sites to appear at any position within the pattern, instead of the middle only [45]. It created a much larger dataset. This increased dataset facilitates the learning of the deep learning method at the cost of increased running time. We utilized the original dataset setting [3, 8, 38, 46]. DiCleave is the current state-of-the-art (SOTA) for this problem with the original dataset setting.

These models suffer several limitations. They rely heavily on complicated feature engineering or opaque deep learning models [38, 46, 45]. It results in a lack of generalizability and a long running time. There is a need to design a simpler model so that it can be easily extended to other prediction tasks on RNA data. One way to analyze sequence data is to transform it into time series data. In response to this, we proposed a multivariate time series classification-based method. Our contributions are shown as follows.

1. To the best of our knowledge, we are the first to frame the prediction of the cleavage sites as a multivariate time series classification problem.

2. We introduced several encoding methods to convert RNA data to time series.

3. We proposed utilizing the base-pair probabilities in the predicted secondary structure for the prediction. To our surprise, this information has been ignored in the existing studies.

4. For computational efficiency, our method achieves a 3.7X to 28.8X speedup compared to the state-of-the-art (SOTA).

5. We conducted perturbation-based experiments. It shows that regions close to the cleavage sites are important for this problem. It is consistent with the existing study [38].

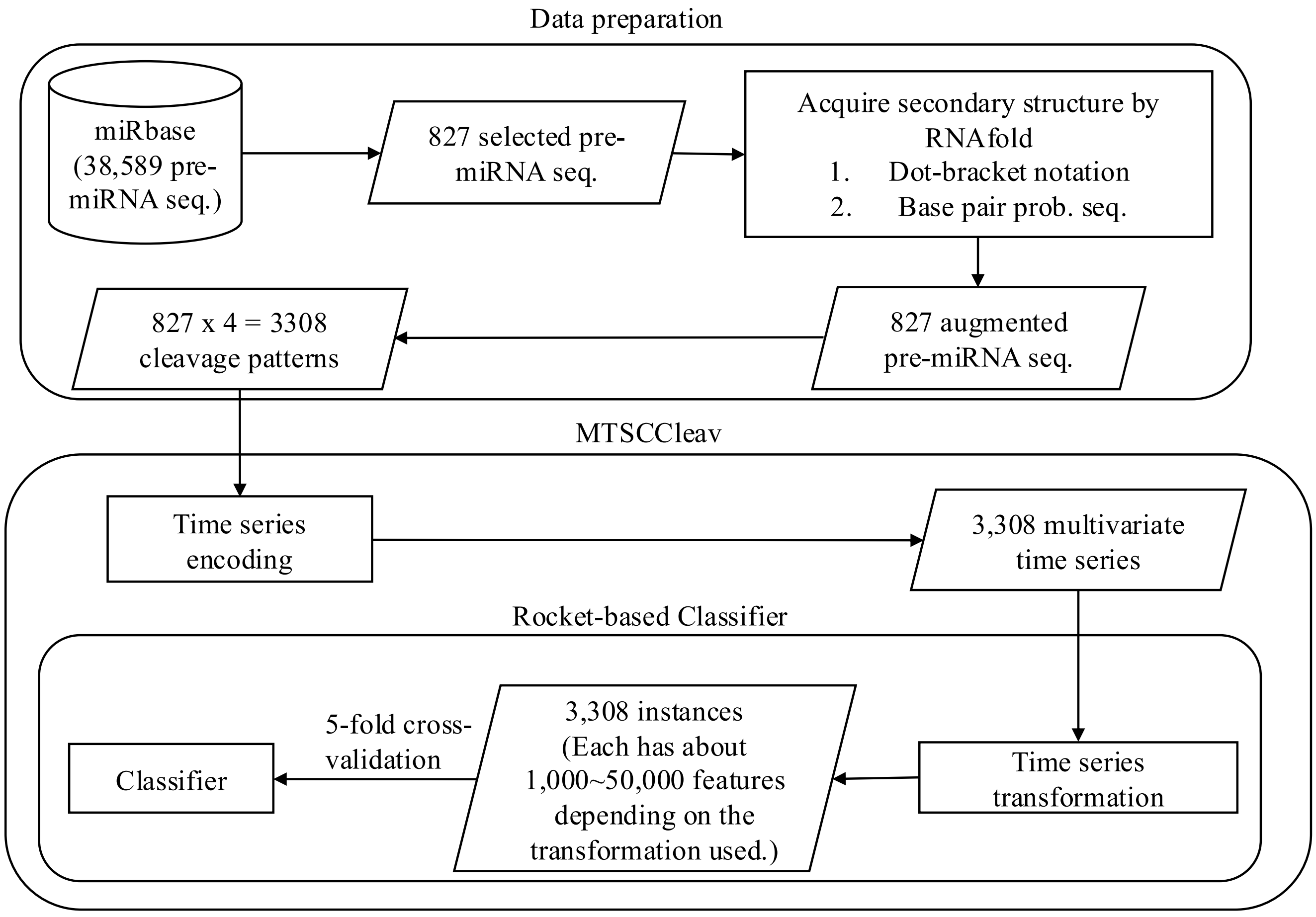
3.2 Methods

The overall pipeline of this study is summarized in Figure 3.2.

3.2.1 Data Preparation

We used miRBase database [24][5]. The database comprises miRNA data from various organisms [63]. The database contains 38,589 miRNA records. Each record refers to an miRNA sequence, along with other properties such as name, accession, organism, and information on its derivative miRNA products. We are interested in pri-miRNA in humans. The derivative miRNA products are the mature miRNAs. The database also annotates the location of the mature miRNA within the original sequence and indicates whether its existence has experimental evidence.

[5]The website is `www.mirbase.org`, and the newest version of the database is Release 22.1 (Accessed on 2025-06-22).



Data preparation
miRbase (38,589 pre-miRNA seq.)
827 selected pre-miRNA seq.
Acquire secondary structure by RNAfold
1. Dot-bracket notation
2. Base pair prob. seq.
827 augmented pre-miRNA seq.
827 x 4 = 3308 cleavage patterns
MTSCCleav
Time series encoding
3,308 multivariate time series
Rocket-based Classifier
Time series transformation
3,308 instances (Each has about 1,000~50,000 features depending on the transformation used.)
5-fold cross-validation
Classifier

Figure 3.2: The overall pipeline of this study. Symbol notations: Cylinder - Dataset, Rectangle - Process, Parallelogram - Input / Output, Rounded Rectangle - Component.

Table 3.1 shows its four representative records. We first selected the records from humans (Homo sapiens). It resulted in 1,917 records. To identify the actual locations of the two cleavage sites in the pri-miRNA sequence supported by experimental evidence, we selected records that have two mature miRNAs resulting from cleavage at the 5p arm and the 3p arm, both of which have experimental support. Hence, only "MI0000060" ("hsa-let-7a-1") would be selected in the table. It would serve as our running example. Its whole sequence is listed in Table 3.2. After the

Accession	Name	Organism	Sequence	Mature miRNA 1	Mature
MI0000001	cel-let-7	Caenorhabditis elegans	$UACAC \cdots UUCGA$	cel-let-7-5p 17:38 experimental	cel-l 6 expe
MI0000060	hsa-let-7a-1	Homo sapiens	$UGGGA \cdots UCCUA$	hsa-let-7a-5p 6:27 experimental	hsa-l 5 expe
MI0000114	hsa-mir-107	Homo sapiens	$CUCUC \cdots ACAGA$	hsa-miR-107 50:72 experimental	
MI0000238	hsa-mir-196a-1	Homo sapiens	$GUGAA \cdots UUCAC$	hsa-miR-196a-5p 7:28 experimental	hsa-miF 4 not exp

Table 3.1: Selected representative records from miRBase. For the last two columns, the first line shows the name, the second line shows its location in the original sequence, and the third line indicates whether its existence has experimental evidence. The selected one is highlighted in **bold**.

selection process, we selected 827 experimental validated pre-miRNA sequences, each with its two mature miRNA products. This formed our dataset.

Sequence	Secondary Structure (In Dot-bracket notation)
1 UGGGA**UGAGGUAGUAGGUUGUAUAGUU** 27 28 UUAGGGUCACACCCACCACUGGGAGAU 54 55 AA**CUAUACAAUCUACUGUCUUUC**CUA 80	1 (((((**.(((((((((((((((((((((** 2 28 UUAGGGUCACACCCACCACUGG 55))**)))))))))))))))))))))**))) 8
Base-pair probabilities sequence (the first 10 bases)	
1 (0.549, 0.946, 0.987, 0.987, 0.904) 5 6 (**0.000**, 0.841, 0.974, 0.981, 0.890) 10	

Table 3.2: The whole sequence of "hsa-let-7a-1" and its predicted secondary structure by RNAfold. The corresponding positions of the two mature miRNAs and the probability of the unpaired "U" are highlighted in **bold**.

Argument the dataset with Secondary Structure information

We leveraged the predicted secondary structure of these sequences to enhance the accuracy of the classification. Recall that a specific three-dimensional (3D) structure is required for DNA, RNA, and protein to perform functions [69]. However, finding these 3D structures using experimental methods such as X-ray crystallography or nuclear magnetic resonance (NMR) is costly and time-consuming. Hence, prediction methods for such 3D structures are necessary and helpful for downstream analysis. However, predicting the 3D structures is challenging. One of the reasons is that there are some "nonconventional" base-pair interactions (e.g., noncanonical and rare A-G) that allow an RNA sequence to fold into a 3D structure, in addition to the (G, U) wobble pair, which is common and functionally important in RNA secondary structures. It makes the search space for prediction much larger than, in the 2D case, the secondary structure. The local structures of the 3D structures, the secondary structures, only focus on the conventional base-pair interactions [5]. Hence, predicting secondary structures is easier and faster. We employed RNAfold from the ViennaRNA Package[6] to predict the secondary structure for a given pri-miRNA S [39]. RNAfold returns the secondary structure in the dot-bracket notation and a matrix of base-pair probabilities. The matrix is a square matrix with the side length $|S|$, where each entry m_{ij} is the probability of base s_i paired up with base s_j. Dot-bracket notation is a way of representing the secondary structure of S. Open parentheses "(" (Close parentheses ")") indicates that the base is paired with a complementary base further (earlier) along in S. Dot "." indicates that the base is unpaired. Equipped with the matrix, we can

[6]The latest stable release is Version 2.7.0 (Accessed on 2025-06-22).

construct the base-pair probability sequence of S. The predicted secondary structure and the base-pair probability sequence of our running example are shown in Table 3.2.

Extract cleavage patterns

The locations of the two mature miRNAs on the whole sequence indicate the probable locations of the two cleavage sites. The 5p cleavage site must be beyond and near the ending location of the 5p mature miRNA. We deemed the immediate bond next to the 5p mature miRNA's ending position the 5p cleavage site, with the knowledge that the actual cleavage site may not be this immediate bond but rather the nearby bonds after it. The same applies to the 3p cleavage site. It is located at the immediate bond before the starting position of the 3p mature miRNA.

For each arm of each whole sequence, we extracted a 14-string[7] with the cleavage site located at the center of the string. The first 7 nt (nucleotide) before the center are highlighted in **bold**. In our running example, it would be "**UAUAGUU**UUAGGU" for the 5p cleavage site and "**GAGAUAA**CUAUACA" for the 3p cleavage site. We refer to these 14-strings as cleavage patterns. We also generate non-cleavage patterns by selecting a 14-string with the center 6 nt away from the corresponding cleavage sites towards the corresponding mature miRNA [8, 38] for each arm of each whole sequence. So, in our running example, the 5p non-cleavage pattern would be "**AGGUUGU**AUAGUUU". The 3p non-cleavage pattern would be "**ACUAUAC**AAUCUAC".

[7]String with length = 14.

In conclusion, for a given pri-miRNA sequence, we can generate two cleavage patterns and two non-cleavage patterns. We call these four patterns simply the "four strings" of a given pri-miRNA. We also call each string a strand. The "four strings" of our running example are listed in Table 3.3.

	5p cleav	5p non-cleav	3p cleav	3p non-cl
Input strand	UAUAGUUUUAGGGU	AGGUUGUAUAGUUU	GAGAUAACUAUACA	ACUAUACAA
Complementary strand	AUAUCAA_____UA	UCUAACAUAUCAA_	C_CUGUUGAUAUGU	UGAUAUGUU

Table 3.3: The first row shows the "four strings" of "hsa-let-7a-1". Their complementary strands are shown in the second row. As a whole, they are referred to as the "eight strings".

We can construct the complementary strand of each of the strands in the "four strings" by finding the corresponding paired base for each of the bases in the input strand by considering the secondary structure information. We use "_" to denote the unpaired base in the complementary strand. For example, in Figure 3.1, "UUAGG" in the 5p cleavage pattern is unpaired, while other bases pair with some bases, the resulting complementary strand is "AUAUCAA_____UA". There is a loop/budge there. We refer to the "four strings" and the four complementary strands together as the "eight strings" of the input pre-miRNA. It is also shown in Table 3.3.

3.2.2 Time Series Encoding

A *time series* $T = t_1, t_2, ..., t_n$ is a sequence of real-valued numbers[8]. A short contiguous region of T is called a subsequence. A *subsequence* $T(i : j) = t_i, t_{i+1}, ..., t_j$

[8]Unless otherwise specified, we denote entries of a time series (e.g., T) using the corresponding lowercase letter (e.g., t).

of a time series T is a shorter time series that starts from position i and ends at position j, where $i < j$.

Strings and time series are temporal sequences. The difference between strings and time series lies in their behavioral attributes [1]. For strings, an entry is a letter from a predefined set called the *alphabet*. For example, the alphabet is $\{A, C, G, T\}$ in the DNA string, while $\{A, C, G, U\}$ in the RNA string. For time series, an entry is a real number. Unlike real numbers, there is no ordering in the alphabet unless some external domain knowledge is introduced.

The study of applying signal processing techniques to genomic data is called "Genomic Signal Processing" (GSP) [42, 6]. In the field of GSP, the time series representations of DNA strings are referred to as DNA numeric representations (DNR). Many DNRs have been proposed. We noted that DNA strings and RNA strings are equivalent from a computational standpoint. Many transformation methods designed for DNA can be applied to RNA by simply substituting T with U. We present nine encoding methods. The relationship among them is shown in Figure 3.3.

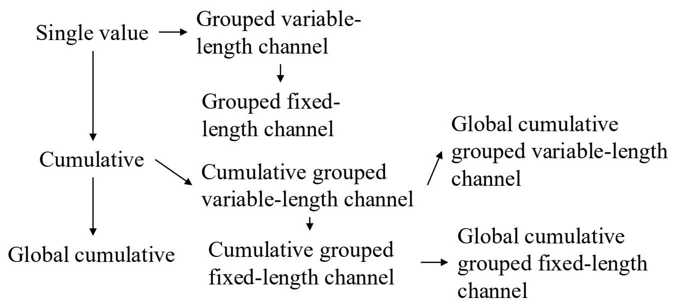

Single value
Grouped variable-length channel
Grouped fixed-length channel
Cumulative
Cumulative grouped variable-length channel
Global cumulative grouped variable-length channel
Global cumulative
Cumulative grouped fixed-length channel
Global cumulative grouped fixed-length channel

Figure 3.3: Relationship of the proposed encoding methods.

Single value versus Cumulative

One of the simple, if not the simplest, encoding is to map the letters into numbers. Domain knowledge can be utilized. This approach is called the "Single value mapping" [48, 13, 11, 67, 42]. One single value is assigned to each of the letters. [27] employed the atomic number of each nucleotide as the transformed values, where $\{G = 78, A = 70, C = 58, T = 66\}$. [47] used electron-ion interaction potential representation (EIIP) as such value, where $\{G = 0.0806, A = 0.1260, C = 0.1340, T = 0.1335\}$. Our goal is to transform the input strand and its complementary strand into time series, aiming to capture the information contained in these sequences and the secondary structure implied by them. We employed the following reasoning to assign the value:

1. We employ the complementary property [4, 11] during encoding. Recall that in the base-pairing rules, G pairs with C to form three hydrogen bonds while A pairs with U[9] to form two hydrogen bonds. G-C pairs are more stable than A-U pairs. G (U) can be regarded as the "inverse" of C (A). We can preserve these base-pairing rules in the encoding by assigning G (A) and C (U) opposite values.

2. G and A have a two-ring structure. They are purines. C and U have a single-ring structure. They are pyrimidines. Hence, we put G and A (C and U) on the same side of the number line with zero in the middle.

3. The lower stability of A-U pairs promotes strand separation, thereby facilitating the unwinding of the miRNA duplex during RISC loading. Regions

[9] In DNA, A pairs with T.

rich in A and U are thus more likely to undergo strand selection and cleavage events. We assigned A (U) with a larger absolute value than G (C) to reflect this functional relevance. It aims to highlight sequence regions with higher cleavage potential.

It results in our baseline transformation method, namely "Single value mapping" as shown in row 1 of Table 3.4. S is the input strand. When we encode S without incorporating the corresponding base-pair probability sequence P, we set $p_i = 1$ for all the entries of P. We use the first ten nucleotides of the complementary strand of the 3p cleav of "hsa-let-7a-1", as shown in Table 3.3 as S in the examples in Table 3.4.

With the assigned value to each nucleotide defined in single-value mapping, we can compute a cumulative sum of those values over time. It captures the aggregated signal by accumulating past events, allowing us to focus on the trend [65, 10]. We named this method as "Cumulative mapping", shown in row 4 of Table 3.4.

Grouped variable-length channel versus Grouped local-length channel

We can transform the input strand into a multivariate time series with two channels using grouped binary encoding, where nucleotides are grouped into (A, U) and (G, C). It releases our third assumption that A (U) has a larger absolute value than G (C). We proposed two variations. The first one allows the output to be variable-length sequences per channel, depending on group-specific occurrences. The second one always returns two resulting sequences of a fixed length. Two variations extended from single value mapping are shown in rows 2 and 3, while those extended from cumulative mapping are shown in rows 5 and 6 in Table 3.4.

	Encoding	Algorithm	Exa $S = C, \_, C, U,$ $P = 0.843, 0.000,$ $0.914, 0.982, 1.$

1	Single value mapping
	[48, 13, 11, 67, 42]
		for $i = 1$ to $\|S\|$:
		$t_i = \begin{cases} 2 \cdot p_i & \text{if } s_i = A \\ 1 \cdot p_i & \text{if } s_i = G \\ -1 \cdot p_i & \text{if } s_i = C \\ -2 \cdot p_i & \text{if } s_i = U \\ 0 & \text{otherwise} \end{cases}$
		return T
			Without base-pair p
			$T = -1, 0, -1, -2$
			
			With base-pair pr
			$T = -0.843, 0.00$
			$0.793, -1.8$
			$1.000, 1.9$

2	Grouped
	variable-length
	channel mapping
		$j = 1, k = 1$
		for $i = 1$ to $\|S\|$:
		$t^1_j = \begin{cases} 1 \cdot p_i & \text{if } s_i = A \\ -1 \cdot p_i & \text{if } s_i = U \\ 0 & \text{otherwise} \end{cases}$
		$t^2_k = \begin{cases} 1 \cdot p_i & \text{if } s_i = G \\ -1 \cdot p_i & \text{if } s_i = C \end{cases}$
		if $(s_i = G)$ or $(s_i = C)$:
		increment k by 1
		else:
		increment j by 1
		return T^1, T^2
			Without base-pair p
			$T^1 = 0, -1,$
			$T^2 = -$
			
			With base-pair pr
			$T^1 = 0.000, -0.807, -0.9$
			$T^2 = -0.843, -$

3	Grouped
	fixed-length
	channel mapping
		for $i = 1$ to $\|S\|$:
		$t^1_i = \begin{cases} 1 \cdot p_i & \text{if } s_i = A \\ -1 \cdot p_i & \text{if } s_i = U \\ 0 & \text{otherwise} \end{cases}$
		$t^2_i = \begin{cases} 1 \cdot p_i & \text{if } s_i = G \\ -1 \cdot p_i & \text{if } s_i = C \\ 0 & \text{otherwise} \end{cases}$
		return T^1, T^2
			Without base-pair p
			$T^1 = 0, 0, 0, -1,$
			$T^2 = -1, 0, -1$
			
			With base-pair pr
			$T^1 = 0.000, 0.00$
			$0.000, -0.9$
			$0.000, 0.99$
			$T^2 = -0.843, 0.0$
			$0.793, 0.0$
			$1.000, 0.$

4	Cumulative mapping
	[65, 10]
		$t_1 = 0$
		for $i = 1$ to $\|S\|$:
		$t_{i+1} = \begin{cases} t_i + 2 \cdot p_i & \text{if } s_i = A \\ t_i + 1 \cdot p_i & \text{if } s_i = G \\ t_i - 1 \cdot p_i & \text{if } s_i = C \\ t_i - 2 \cdot p_i & \text{if } s_i = U \\ t_i & \text{otherwise} \end{cases}$
		return T // $\|T\| = \|S\| + 1$
			Without base-pair p
			$T = 0, -1, -1, -2, -4,$
			
			With base-pair pr
			$T = 0.000, -0.84$
			$-3.265, -2.471$
			$-5.264, -3$

5	Cumulative grouped variable-length	$t^1_1 = 0, t^2_1 = 0$ $j = 1, k = 1$ for $i = 1$ to $\|S\|$: $t^1_{j+1} = \begin{cases} t^1_j + 1 \cdot p_i & \text{if } s_i = A \\ t^1_j - 1 \cdot p_i & \text{if } s_i = U \\ t^1_j & \text{if } s_i = \_ \end{cases}$	Without base-pair p $T^1 = 0, -1, -$ $T^2 = 0, -$ With base-pair pr

Global cumulative versus Local Cumulative

In cumulative mapping and its variations, we can choose where to start the accumulation. For a given subsequence S' of the whole sequence S, accumulation can start from the beginning of S even if only S' is used downstream. It can also begin just at the start of the S'. The first one preserves the global context. It can be useful when previous nucleotides (those before S') influence later interpretation. The second one focuses solely on local history in S', ignoring global history. It is helpful if the previous nucleotides do not affect the chemical property of S'.

Consider $T = 0, -1, ..., -6$ of the input string S in "Cumulative mapping" in Table 3.4, which accumulates from 0. S is the suffix with length $= 10$ of the constructed complementary strand of $S(1 : 63)$ in Figure 3.1. If we start the accumulation from the first entry of the constructed complementary strand instead, it will yield a different result. Suppose that the last entry of the time series encoded in the cumulative mapping of the constructed complementary strand is -8, the time series encoded in the "Global cumulative mapping" for S would accumulate from -8 instead of 0. The result is $T = -8, -9, ..., -14$. Note that it has the same trend as the original T. This "Global cumulative" concept can be applied to every cumulative-based method, as shown in Figure 3.3.

Incorporating base-pair probabilities

We can incorporate the base-pair probabilities P in the encoding by thinking of it as the weight or confidence p_i in the value assignment of each nucleotide s_i. It is implemented by multiplying the base-pair probability p_i of the nucleotide s_i with the assigned value of the kind of nucleotide of s_i during encoding, as shown in Table 3.4.

Transforming the secondary structure into time series

We can transform the secondary structure in the dot-bracket notation into a time series by "Single value mapping", where "(" maps to 1, "." maps to 0, and ")" maps to -1.

3.2.3 Time series classification

In univariate time series classification, an instance in the dataset consists of a time series $x = x_1, x_2, ..., x_m$ with m observations and a discrete class label y, which takes c possible values [7, 52]. If $c = 2$, we refer to binary classification. If $c > 2$, we refer to multi-class classification. In multivariate time series classification, the time series is not a single sequence but a list of sequences. Each sequence is called a channel. There are many classifiers defined for time series data, including distance-based, feature-based, interval-based, shapelet-based, dictionary-based, convolution-based, and deep learning-based classifiers. Additionally, two or more of the above approaches can be combined, resulting in hybrid approaches [44, 52, 7]. We employed convolution-based classifiers due to their simplicity and accuracy.

Convolution-based classifiers

Convolution-based classifiers first use randomly parameterized kernels to perform convolutions on the original time series T. A kernel is referred to as parameterized because its behavior is governed by a set of parameters, which will be discussed in detail later. Convolution is an operation to transform T to another time series M, where M is called the activation map. Its entry M_i is calculated by applying a kernel ω with length l to T at position i, defined as follows:

$$M_i = T(i : i + l - 1) * \omega = \sum_{j=0}^{l-1} t_{i+j} \cdot \omega_{1+j}$$

To note, $|T(i : i + l - 1)| = |\omega| = l$. Entries M_i's are calculated by sliding ω across T and computing a dot product. Additionally, although the original paper [16] used the term "convolution" to refer to the above operation, "cross-correlation" may be a more suitable term for this operation. Recall T with length m has $(m - l + 1)$ sliding windows of length l, given that the increment is 1[10], which defines the length of M.

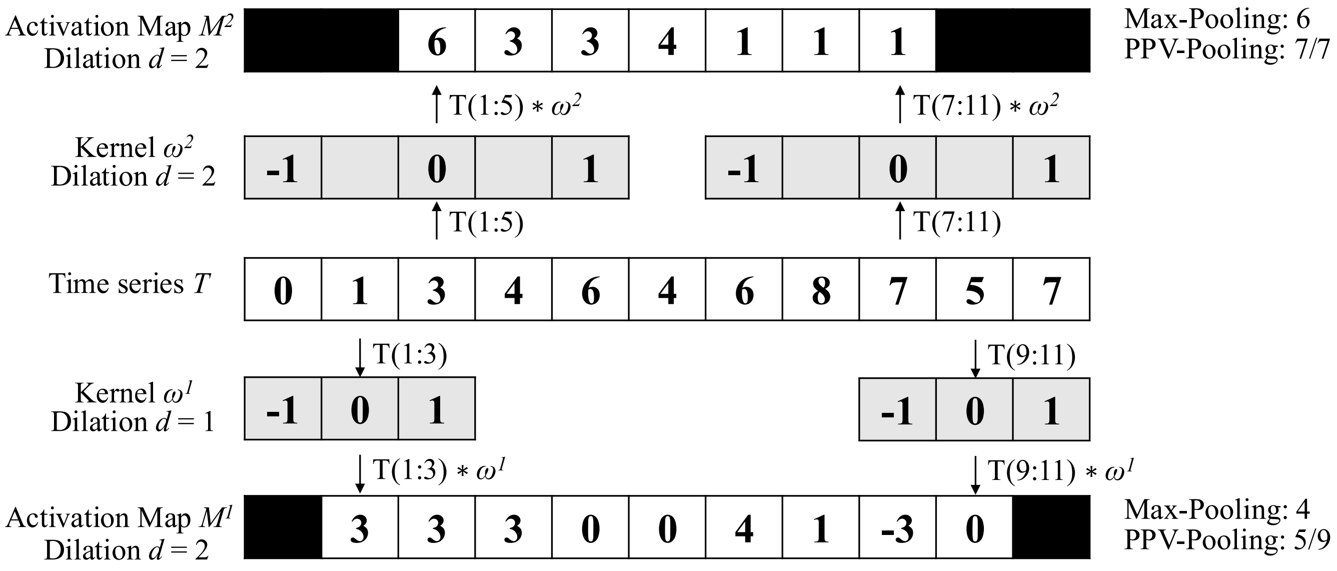


Figure 3.4: Features generation in the transformation

[10]One step to the right per time.

Figure 3.4 shows two kernels ω^1 and ω^2 with lengths 3 and 5, respectively. Each of which performs a convolution with T and returns two activation maps, M^1 and M^2, respectively. For example, $M_1^1 = T(1:3) * \omega^1 = 3$. By sliding ω^1 one time stamp at a time, an activation map M^1 with length $= (m-l+1) = 11-3+1 = 9$ is obtained. Then, pooling operations, such as the maximum (MAX) and proportion of positive values (PPV), are applied on M^1 to derive the summary features. In Figure 3.4, MAX and PPV are applied on M^1 and M^2. The summary features of M^1 are 4 and 5/9, which correspond to MAX and PPV, respectively. Dilation refers to a method that enables a kernel to cover a larger portion by creating empty spaces between entries in the kernel. The dilation d of ω^2 is 2. It introduces a gap of 1 in every two values of ω^2.

The most popular convolution-based approach is the Random Convolutional Kernel Transform (ROCKET) [16]. It generates a large number of randomly parameterized kernels, ranging from thousands to tens of thousands. The kernel's parameters include length, weights (the entries inside the kernel), bias (the value added to the result of the convolution operation), and dilation. Additionally, padding can be applied to T at the start and end, ensuring M has the same length as the input. To note, T, M_1, and M_2 in Figure 3.4 have different lengths. The summary statistics of the activation map are obtained through two pooling operations: MAX and PPV. Hence, for k kernels, the transformed data has $2k$ features. The default value of k is 10,000.

There are two extensions of ROCKET. They are MiniROCKET [17] and MultiROCKET [58]. MiniROCKET removes unnecessary operations and many of the random components in the definition of kernels used by ROCKET. It speeds up Rocket by over an order of magnitude with no significant difference in accuracy, making the classifier almost deterministic. For example, the kernel length is fixed, and only two weight values are used. Only PPV is used for the summary statistics. MultiROCKET is extended from MiniROCKET. The main improvement of it is to extract features from first-order differences as defined in Table 3.5 and add three new pooling operations [58]. The three added operations are mean of positive values (MPV), mean of indices of positive values (MIPV) and longest stretch of positive values (LSPV).

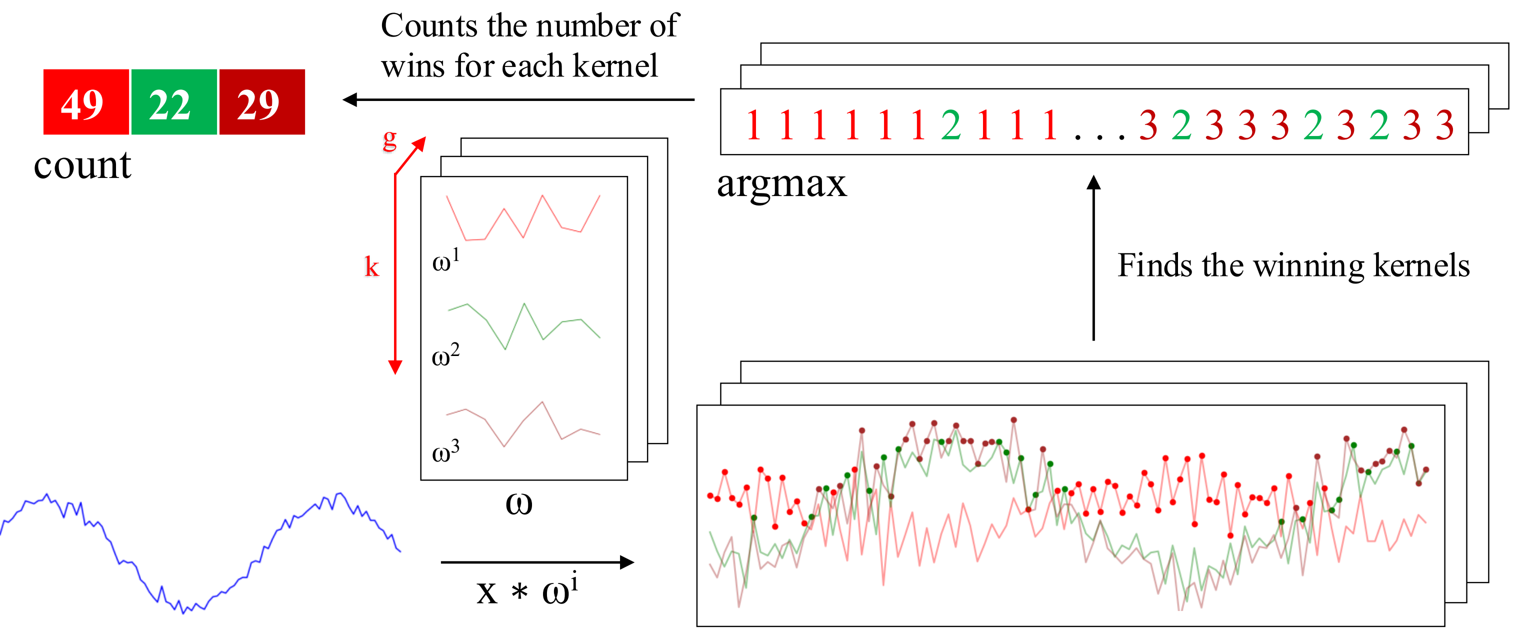


Figure 3.5: Convolutions of HYDRA for each input time series with a set of random kernels w, organized into g groups with k kernels each.

The HYbrid Dictionary-ROCKET Architecture (Hydra) combines dictionary-based and convolution-based models [18]. Similar to ROCKET-based classifiers, it uses random kernels to extract features from the input time series. But it groups the kernels into g groups of k kernels each, as shown in Figure 3.5. Each time series is passed through all the groups. For each group of kernels, we slide them across T and compute the dot product at each timestamp. Recall that the dot product of two input vectors (x and w_i) has the maximum value when the two vectors align in the same direction and the minimum value when they are oriented in opposite directions. We record the kernel that best matches the subsequence of T at each timestamp in each group (i.e., argmax). We refer to these kernels as the winning kernels. This results in a k-dimensional count vector for each of the g groups, where $k = 3$ in Figure 3.5. This results in a total of $g \times k$ features, with default values of $g = 64$ and $k = 8$ It uses a total of $k \times g = 512$ kernels per dilation. In addition to recording the kernel with the maximum response, we can also record the kernel with the minimum response, knowing that this kernel will be the best match with the "inverted" subsequence of T. Hydra is applied to both the original time series and its first-order differences. Hydra generated approximately 1000 features for each instance in our dataset. [18] found that it can improve the accuracy by concatenating features generated from Hydra with those from MultiRocket. This classifier is called MultiROCKET-Hydra.

These five classifiers share the same simple design pattern. It involves the overproduction of features followed by a selection strategy. A large number of features (1,000 $\sim$ 50,000) are generated for each instance. The features are then fed into a simple linear classifier. It determines which features are most useful and returns the final classification result. A ridge classifier is used in this study. It is a linear classifier that extends ridge regression to classification tasks by applying a threshold to the predicted values. It uses L2 regularization to prevent overfitting. The regularization strength is selected by internal cross-validation. A Ridge classifier is suggested for small datasets, as in our case, while a logistic regression classifier is suggested for large datasets [44].

While these five classifiers are often referred to as classifiers [44], they are technically time series transformation methods for generating features that are then fed to a downstream classifier. The comparison of them is shown in Table 3.5. For MiniROCKET and MultiROCKET, the bias is determined from the convolution output, and the dilation depends on the length of the input time series [17, 58]. The main differences among ROCKET-based classifiers lie in how the summary features are generated. The generation of the summary features depends on:

1. Kernels, which are defined based on the parameters, which consist of kernel length, kernel weights, bias, and dilation.

2. The way that padding applies to T, which leads to activation maps with different lengths.

3. The pooling operations, which are used in extracting features on the activation map.

	ROCKET	MiniROCKET	MultiROCKET	Hy
kernel length	{7, 9, 11}	9	9	9
kernel weights	$\mathcal{N}(0,1)$	{-1, 2}	{-1, 2}	$\mathcal{N}($
bias	$\mathcal{U}(0,1)$	from output	from output	nc
dilation	random	fixed (input-relative)	fixed (input-relative)	ran
padding	random	fixed	fixed	alw
pooling operations	MAX, PPV	PPV	PPV, MPV, MIPV, LSPV	Response per
1st order difference	no	no	yes	y
feature vector size	20k	10k	50k	relative

Table 3.5: Comparison of rocket-based classifiers [44]. $\mathcal{N}(0,1)$: a standard normal distribution, $U(0,1)$: a uniform distribution between 0 and 1, 1st order difference: $\Delta T = t_2 - t_1, t_3 - t_2, ..., t_n - t_{n-1}$.

3.2.4 Evaluation metrics

To evaluate the performance of our time series-based classification (MTSC) model, we adopted five standard classification metrics. They are Accuracy (Acc), Specificity (Sp), Sensitivity (Sn), F1 score (F1), and Matthews Correlation Coefficient (MCC) [41].

$$
\begin{aligned}
Acc &= \frac{TP + TN}{TP + TN + FP + FN} \\
Sp &= \frac{TN}{TN + FP} \\
Sn &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times TP}{2 \times TP + FP + FN} \\
MCC &= \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}
\end{aligned}
$$

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives, and false negatives, respectively.

To extend a binary metric to multi-class problems, we can treat the data as a collection of binary problems, one for each class. One class is treated as positive while the other classes are treated as negative. Then, the multi-class metrics can be obtained by averaging binary metric calculations across the set of classes. There are different ways of doing the averaging. Here, we adopted a macro-averaging approach. It treats each class equally and calculates the mean of the binary metrics. To use MCC in the multiclass case, it can be defined in terms of a confusion matrix C for K classes, where $C_{i,j}$ is the number of observations that are actually in class i and predicted to be in class j [23].

$$MCC_{multi} = \frac{c \times s - \sum_k^K p_k \times t_k}{\sqrt{(s^2 - \sum_k^K p_k^2) \times (s^2 - \sum_k^K t_k^2)}}$$

where $t_k = \sum_i^K C_{i,k}$ (denoting the number of times class k actually occurred), $p_k = \sum_i^K C_{k,i}$ (denoting the number of times class k was predicted), $c = \sum_k^K C_{k,k}$ (denoting the total number of samples correctly predicted) and $s = \sum_i^K \sum_j^K C_{i,j}$ (denoting the total number of samples).

3.3 Results

The code implementing our method is available at `https://github.com/cyuab/time-series-classification-cleavage`. The dataset of this study is available at `https://www.mirbase.org`.

In all experiments, the models were trained and tested using 5-fold cross-validation. We retrieved 827 empirically validated sequences of pre-miRNAs. There are 5p arm and 3p arm in each sequence. For each arm, we defined a cleavage pattern and a non-cleavage pattern. Three datasets, namely "5p arm", "3p arm", and "multi-class" were constructed by these patterns. We refer to the cleavage patterns as positive instances and the non-cleavage patterns as negative instances. The 5p arm dataset comprises 827 positive instances and an equal number of negative instances. The 5p arm and 3p arm datasets are binary-class datasets. The multi-class dataset comprises all patterns from both the 5p arm and the 3p arm. There are 827 “5p” instances[11], 827 “3p” instances, and 1,654 negative instances.

For every fold in 5-fold cross-validation, the dataset was divided into a training set and a test set with sizes of 80% and 20% of the whole dataset, respectively. We kept the class distribution approximately the same in each fold, since it is in the original dataset. In each fold derived from the 5p arm and 3p arm datasets, the training set has a size of 1,323, and the test set has a size of 331. In each fold derived from the multi-class dataset, the training set has a size of 2,262, and the test set has a size of 662. We reported the average of the five classification metrics.

The ROCKET-based classifiers require all channels in the multivariate time series to have equal length. We applied padding to the shorter channels using the constant value 100, which does not appear in the original time series. It ensures the padding does not introduce ambiguity or interfere with the semantic meaning of the encoded nucleotide signals.

3.3.1 Channel ablation study

We utilized three types of data as the input features for each instance. They are (1) the RNA sequence, which consists of the primary strand and its complementary strand, (2) the secondary structure information, and (3) the base-pair probability sequence. To input the data into our time series-based classifiers, we converted them into multivariate time series. The primary strand and its complementary strand are each encoded into one or two channels, using the encoding methods in Table 3.4. For example, single value mapping encodes a strand in one channel, while grouped variable-length channel mapping encodes in two channels. The

[11]Cleavage patterns from the 5p arm.

secondary structure information is converted into a univariate time series. The base-pair probability sequence is already in numerical form and does not require further transformation. It can be used either as a standalone channel or incorporated into the encoding of the complementary strand. We performed a channel ablation study to determine the most informative combination of the above channels.

We referred to the multivariate time series that consists of the channels from the RNA sequence only as the baseline setting. We added the other channels to this baseline. It leads to the following configurations (cfgs):

1. (cfg 1) Baseline: Time series derived only from the RNA sequence.
2. (cfg 2) Baseline + Secondary structure: Baseline + time series representation of the secondary structure.
3. (cfg 3) Baseline + Base-pair probability (Standalone): Baseline + the base-pair probability sequence as a standalone channel.
4. (cfg 4) Baseline + Base-Pair probability (Incorporated): Baseline with the base-pair probability sequence incorporated into the encoding of the complementary strand.

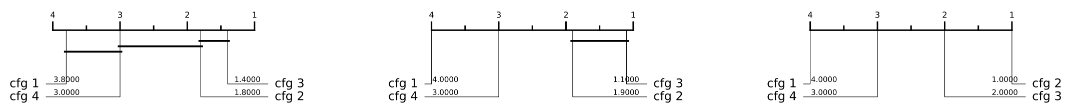
We used single value mapping as the encoding method. Table 3.6 shows the result. From the table, we can see that the addition of secondary structure, base-pair probability as a standalone channel, and base-pair probability incorporated in the encoding of the complementary strand can improve the performance. We plotted the critical difference (CD) diagram as shown in Figure 3.6 to visualize Table 3.6 to make the performances of different combinations more obvious. In

	Classifier	5p arm						
		Acc	Sp	Sn	F1	MCC	Acc	Sp
Baseline (cfg 1)	ROCKET	0.781	0.743	0.819	0.789	0.563	0.790	0.77
	MiniROCKET	0.755	0.728	0.782	0.762	0.512	0.788	0.78
	MultiROCKET	0.784	0.767	0.801	0.787	0.569	0.803	0.79
	Hydra	0.830	0.800	0.860	0.835	0.663	0.808	0.79
	MultiROCKET-Hydra	0.796	0.778	0.815	0.800	0.594	0.807	0.76
Baseline + Secondary Structre (cfg 2)	ROCKET	**0.847**	**0.832**	0.862	**0.849**	**0.695**	**0.855**	0.84
	MiniROCKET	0.825	0.807	0.843	0.827	0.652	0.822	0.80
	MultiROCKET	0.812	0.803	0.822	0.814	0.626	0.824	0.80
	Hydra	0.845	0.816	**0.873**	**0.849**	0.691	0.846	0.81
	MultiROCKET-Hydra	0.817	0.809	0.826	0.819	0.635	0.825	0.81
Baseline + Base-pair probability (Standalone) (cfg 3)	ROCKET	0.842	0.828	0.855	0.844	0.684	**0.855**	**0.85**
	MiniROCKET	0.817	0.820	0.814	0.816	0.634	0.836	0.83
	MultiROCKET	0.822	0.813	0.832	0.824	0.645	0.825	0.83
	Hydra	0.846	0.827	0.865	**0.849**	0.693	0.851	0.84
	MultiROCKET-Hydra	0.822	0.809	0.834	0.824	0.644	0.835	0.84
Baseline + Base-pair probability (Incorporated) (cfg 4)	ROCKET	0.799	0.771	0.827	0.805	0.600	0.809	0.78
	MiniROCKET	0.776	0.756	0.797	0.781	0.554	0.801	0.80
	MultiROCKET	0.814	0.801	0.828	0.817	0.630	0.816	0.81
	Hydra	0.822	0.787	0.857	0.828	0.647	0.834	0.82
	MultiROCKET-Hydra	0.814	0.802	0.820	0.817	0.629	0.820	0.82

Table 3.6: Channel ablation study. The best results are highlighted in **bold**.

CD diagrams, lower-ranked methods (toward the right) are better. A horizontal bar connecting combinations indicates no statistically significant difference.

From Figure 3.6, we can see that including time series derived from secondary structure information and base-pair probability as a separate channel can significantly improve the performance of the classifiers. Incorporating the base-pair probability sequence in the time series encoding of the complementary strand can also improve the classifier, but to a minor degree compared to serving as a standalone channel. In our downstream analysis, we adopted the combination of RNA



(a) 5p arm (b) 3p arm (c) multi-class

Figure 3.6: CD diagrams of channel ablation study.

sequence time series, secondary structure time series, and base-pair probability time series as our multivariate time series input, with 4 to 6 channels, depending on the encoding used.

3.3.2 Predictive performance

The experiment was conducted on three datasets: the 5p arm, the 3p arm, and the multi-class datasets. Recall that we have nine encoding methods and five ROCKET-based classifiers. It results in 45 combinations of encoding methods and classifiers.

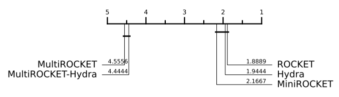
The result is shown in Table 3.7. The best combination of encoding method and classifier is shown in Table 3.8. For the 5p arm dataset, the best combination is "Global Cumulative grouped fixed-length channel mapping + ROCKET". For all five classification metrics, it outperforms the state-of-the-art (SOTA) method, DiCleave. For the 3p arm dataset, the best combination is "Global Cumulative grouped fixed-length channel mapping + ROCKET". Out of the five classification metrics, it outperforms DiCleave, except in specificity. For the multi-class dataset, the best combination is "Global Cumulative grouped fixed-length channel mapping + ROCKET". For all five classification metrics, it outperforms DiCleave. Note that for the 3p arm and the multi-class datasets, the combination of "Cumulative grouped fixed-length channel mapping + ROKCET" also attains the best result.

	Classifier	5p arm						
		Acc	Sp	Sn	F1	MCC	Acc	
Single value mapping (enc 1)	ROCKET	0.849	0.842	0.857	0.851	0.699	0.863	0.
	MiniROCKET	0.823	0.809	0.837	0.825	0.647	0.823	0.
	MultiROCKET	0.821	0.802	0.840	0.824	0.643	0.839	0.
	Hydra	0.843	0.820	0.867	0.847	0.688	0.838	0.
	MultiROCKET-Hydra	0.820	0.803	0.837	0.823	0.640	0.840	0.
Grouped variable-length channel mapping (enc 2)	ROCKET	0.835	0.826	0.844	0.836	0.670	0.855	0.
	MiniROCKET	0.843	0.833	0.853	0.844	0.686	0.831	0.
	MultiROCKET	0.819	0.809	0.828	0.820	0.638	0.817	0.
	Hydra	0.825	0.780	0.869	0.832	0.653	0.811	0.
	MultiROCKET-Hydra	0.818	0.814	0.822	0.819	0.636	0.831	0.
Grouped fixed-length channel mapping (enc 3)	ROCKET	0.851	0.843	0.859	0.852	0.702	0.863	0.
	MiniROCKET	0.844	0.836	0.853	0.845	0.689	0.840	0.
	MultiROCKET	0.831	0.815	0.848	0.834	0.663	0.824	0.
	Hydra	0.848	0.816	0.880	0.853	0.699	0.862	0.
	MultiROCKET-Hydra	0.836	0.813	0.859	0.839	0.672	0.833	0.
Cumulative mapping (enc 4)	ROCKET	0.850	0.834	0.866	0.852	0.701	0.863	0.
	MiniROCKET	0.840	0.821	0.860	0.843	0.682	0.840	0.
	MultiROCKET	0.822	0.809	0.834	0.824	0.644	0.832	0.
	Hydra	0.848	0.819	0.878	0.853	0.698	0.853	0.
	MultiROCKET-Hydra	0.824	0.811	0.856	0.825	0.647	0.838	0.
Cumulative grouped variable-length channel mapping (enc 5)	ROCKET	0.843	0.821	0.866	0.847	0.688	0.856	0.
	MiniROCKET	0.845	0.826	0.865	0.848	0.691	0.836	0.
	MultiROCKET	0.826	0.814	0.838	0.828	0.653	0.815	0.
	Hydra	0.850	0.819	0.880	0.854	0.701	0.834	0.
	MultiROCKET-Hydra	0.824	0.810	0.838	0.826	0.649	0.833	0.
Cumulative grouped fixed-length channel mapping (enc 6)	ROCKET	0.856	0.836	0.876	0.858	0.712	**0.870**	**0.**
	MiniROCKET	0.856	0.837	0.874	0.858	0.712	0.842	0.
	MultiROCKET	0.820	0.802	0.839	0.824	0.642	0.798	0.
	Hydra	0.850	0.814	0.885	0.855	0.701	0.855	0.
	MultiROCKET-Hydra	0.820	0.801	0.839	0.823	0.641	0.807	0.
Global Cumulative mapping (enc 7)	ROCKET	0.850	0.834	0.866	0.852	0.701	0.863	0.
	MiniROCKET	0.847	0.832	0.862	0.849	0.695	0.848	0.
	MultiROCKET	0.827	0.819	0.834	0.828	0.653	0.847	0.
	Hydra	0.851	0.821	0.880	0.855	0.703	0.861	0.
	MultiROCKET-Hydra	0.829	0.823	0.834	0.830	0.658	0.843	0.
Global Cumulative grouped variable-length channel mapping (enc 8)	ROCKET	0.840	0.814	0.867	0.844	0.682	0.853	0.
	MiniROCKET	0.848	0.834	0.862	0.850	0.697	0.841	0.
	MultiROCKET	0.834	0.828	0.839	0.834	0.668	0.831	0.
	Hydra	**0.857**	0.821	**0.894**	**0.862**	**0.717**	0.822	0.
	MultiROCKET-Hydra	0.837	0.834	0.839	0.837	0.674	0.834	0.
Global Cumulative grouped fixed-length channel mapping (enc 9)	ROCKET	0.856	0.836	0.876	0.858	0.712	**0.870**	**0.**
	MiniROCKET	**0.857**	**0.845**	0.870	0.859	0.715	0.840	0.
	MultiROCKET	0.829	0.825	0.833	0.830	0.658	0.820	0.
	Hydra	0.856	0.817	**0.894**	0.861	0.713	0.859	0.
	MultiROCKET-Hydra	0.829	0.824	0.834	0.830	0.658	0.822	0.

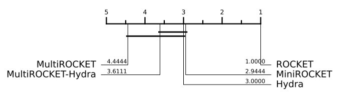
Table 3.7: Performance on the 45 combinations between encoding methods and the ROCKET-based classifiers. The best results are highlighted in **bold**.

Dataset	Methods	Acc	Sp	Sn	F1	MCC	Time (s)
5p arm	enc 9 + MiniROCKET	**0.857**	**0.845**	**0.870**	**0.859**	**0.715**	**0.787**
	DiCleave	0.818	0.790	0.846	0.822	0.653	21.249
3p arm	enc 9 + ROCKET	**0.870**	0.861	**0.879**	**0.871**	**0.741**	4.311
	enc 7 + MiniROCKET	0.848	0.839	0.857	0.850	0.697	**0.989**
	DiCleave	0.854	**0.891**	0.817	0.847	0.715	15.919
multi-class	enc 9 + ROCKET	**0.863**	**0.921**	**0.852**	**0.860**	**0.780**	12.208
	enc 3 + MiniROCKET	0.851	0.915	0.844	0.849	0.760	**4.550**
	DiCleave	0.820	0.895	0.804	0.815	0.710	131.151

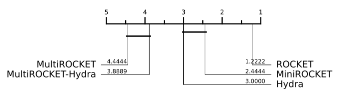
Table 3.8: Comparative analysis between MTSCCleav with the best combination of the encoding method and classifier, with the SOTA, DiCleave, on the three datasets. The best results of using MiniROCKET have also been shown to compare the computational efficiency. The best results are highlighted in **bold**.



(a) 5p arm

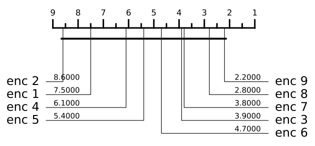


(b) 3p arm

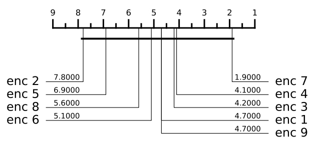


(c) multi-class

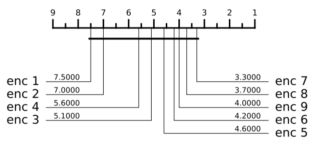
Figure 3.7: CD diagrams to compare different classifiers.



(a) 5p arm



(b) 3p arm



(c) multi-class

Figure 3.8: CD diagrams to compare different encoding methods.

To summarize Table 3.7, we plot the CD diagrams for finding the best classifier, as shown in Figure 3.7, and the best encoding method, as shown in Figure 3.8.

3.3.3 Running time analysis

To compare the computational efficiency of MTSCCleav and DiCleave, we conducted a comparative analysis of their running times. For DiCleave, we employed the code from its supporting website[12], without any modifications. All experiments were conducted on the same machine (a personal laptop equipped with an Apple M1 Pro chip and 16 GB of memory) and using the same splits of the training and test datasets under 5-fold cross-validation to ensure fairness. The reported running times are the averages of the five runs. The timing results were measured from the training phase to the return of the five classification metrics. The result is shown in Table 3.8. MiniROCKET is the most computationally efficient of the five rocket-based classifiers. We also included its best result, along with the corresponding encoding method, even though this combination may not be the best overall.

MTSCCleav demonstrated a significant advantage in computational efficiency, achieving an average 27.0X, 3.7X, and 10.7X speedup over DiCleave, for the 5p arm, 3p arm, and multi-class datasets, respectively. If we consider using the MiniROCKET in the case of 3p arm and multi-class datasets, it achieves 16.1X and 28.8X speedup. To note, in the case of the 3p arm dataset, the performance of MiniROCKET is only slightly worse than DiCleave. In the case of the multi-class dataset, even the performance of MiniROCKET is better than DiCleave.

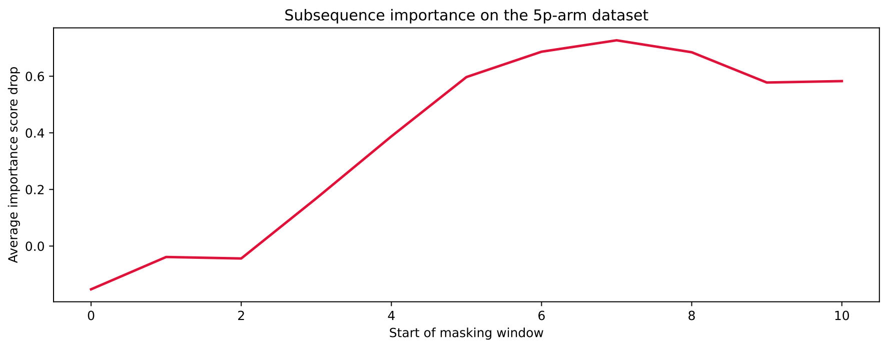
[12] https://github.com/MGuard0303/DiCleave (Accessed on: 2025-07-13).

DiCleave is a deep learning-based method that requires substantial time for model inference, while MTSCleav leverages efficient ROCKET-based classifiers. This significant reduction in runtime makes MTSCCleav more suitable for large-scale data and real-time applications.

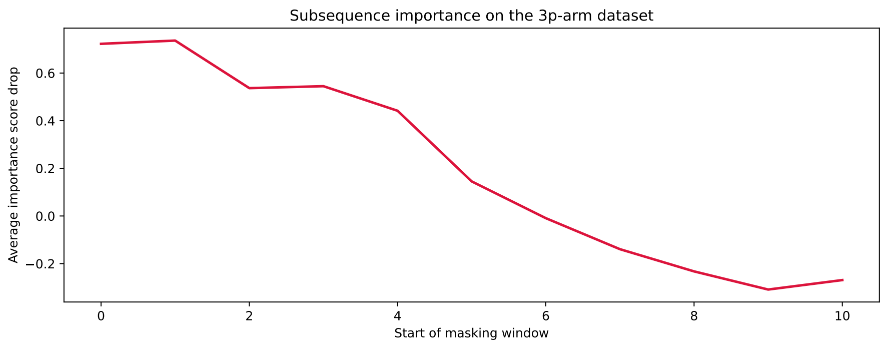
3.3.4 Subsequence importance

To evaluate the sensitivity of MTSCCleav to subsequences of the input, we conducted a perturbation experiment to evaluate the importance of subsequences based on masking windows. The goal of this experiment is to identify which subsequences of the entire time series are critical for classification. We examine how various modifications to the original input impact model performance. It suggests which features are essential for classification.

The model was trained on the original training dataset. For each instance in the test dataset, we measure its original score and the masked score. We slid a masking window w with a fixed length over the input time series T. $|w|$ was set to 4. For each window position $i \in \{1, 2, ..., |T| - |w| + 1\}$, we masked all entries across all the channels of T within the window. Hence, we removed or hid that portion of information from the model during inference. The changes in classification performance in terms of accuracy relative to the unmasked original score of each i are recorded. Intuitively, if the information of a subsequence is critical for the classification, the masking of this subsequence would lead to a great drop in classification performance. We aggregated the importance score across the test dataset.



(a) 5p arm



(b) 3p arm

Figure 3.9: Results of the perturbation experiment.

The result is shown in Figure 3.9. For the encoding methods, we cannot use the methods derived from the cumulative mapping because the accumulation would leak information from the masked region. We adopted "Grouped fixed-length channel mapping" as the encoding method and ROCKET as the classifier. "Grouped fixed-length channel mapping" is the best encoding, other than the methods derived from the cumulative mapping, in all datasets, as shown in Figure 3.8. ROCKET is the best classifier, as shown in Figure 3.7..

In the 5p arm dataset, we found that masking subsequences at the tailing part caused a significant drop in the importance score, as shown in Figure 3.9 (a).

In the 3p arm dataset, we found that masking subsequences at the leading part caused a significant drop in the importance score, as shown in Figure 3.9 (b).

3.3.5 Summary

Our method achieves better or comparable predictive results and a 3.7X to 28.8X speedup compared to the state-of-the-art (SOTA).

3.4 Discussion

The channel ablation study reveals that the involvement of the time series derived from the secondary structure can improve accuracy. It suggests the importance of RNA folding in dicer processing. Futhermore, we found that the base-pair probability sequence of the secondary structure can also enhance accuracy. To the best of our knowledge, it is a novel application of the base-pair probability sequence. Experiments show that using the probability sequence as an additional channel can enhance accuracy more than incorporating it in the encoding. It is likely because keeping it as an additional channel can preserve more information, of both the probability sequence itself and the complementary strand.

Out of the three datasets, the best classifier is ROCKET. The ranking of the five classifiers by performance, starting from the best, is as follows: ROCKET, Hydra, MiniROCKET, MultiROCKET-Hydra, and MultiROCKET. It indicates that the features created from the pooling operations that are only in MultiROCKET but not in MiniROCKET, confuse the final classifier. They are mean of positive values (MPV), mean of indices of positive values (MIPV) and longest stretch of positive values (LSPV) [58]. In contrast, the pooling operator that is only present in

ROCKET but not in MiniROCKET, enhances the classification performance. It is maximum (MAX).

For the encoding methods, we have the following observations. Fixed-length grouped channel mappings outperform variable-length counterparts with one exception in the multi-class dataset, likely because fixed-length schemes better preserve the original positional information of nucleotides within the sequence. Global cumulative methods consistently yield better performance than local cumulative methods. It suggests that the upstream information of the cleavage pattern plays a critical role in identifying cleavage sites. Cumulative-based encodings perform better than single-value mappings, with one exception in the 3p dataset, suggesting that the accumulated nucleotide signal is more informative for cleavage site prediction than the local or isolated presence of nucleotides. In the 5p arm dataset, encoding RNA sequence in two channels appears to worsen the result. This suggests that the 5p arm dataset and the 3p arm dataset need different nucleotide grouping methods for the encoding.

One limitation of DiCleave is overfitting during training because of the relatively small size of the dataset [46]. DiCleave is a deep learning-based method. Deep learning models typically require a large amount of training data to generalize effectively. They are data-hungry. In contrast, MTSCCleav leverages ROCKET-based methods for the classification. They rely on random convolutional feature extraction followed by a simple linear classifier. The Ridge classifier used in this study is less data-hungry compared to deep learning methods due to its use of L2 regularization and the simplicity of its linear model nature. It allows ROCKET-based classifiers, and hence MTSCCleav, to maintain strong predictive performance even in settings with a relatively small dataset size.

The subsequence importance reveals some connections between RNA secondary structure and human dicer cleavage site prediction. The perturbation experiment shows that the leading part of 5p arm and the tailing part of 3p arm are important for the classification. These parts are close to the center of the RNA secondary structure of pre-miRNA. It indicates that the center region is more crucial for human dicer cleavage site prediction. It is consistent with the previous study [38].

3.5 Conclusions

We proposed an accurate, fast, and simple multivariate time series classification (MTSC)-based method, termed MTSCCleav, for predicting human dicer cleavage sites. Base-pair probability sequences of the secondary structures have also been leveraged in the classification. MTSCCleav consists of three parts: time series encoding, time series transformation, and classification. ROCKET-based methods were used for time series transformation. Ridge Classifier was used for classification. For the computational experiments, we evaluated nine time series encoding methods in conjunction with five time series transformation methods. MTSCCleav outperformed the SOTA method in all five evaluation metrics for the 5p-arm and multi-class datasets, and four of the metrics for the 3p-arm dataset. In terms of computational efficiency, MTSCCleav with the optimal setting achieved an average 3.7X to 27.0X speedup over the SOTA method on the three datasets. With the use of a less accurate but faster time series classification method, MTSCCleav achieved an average speedup of 16.1X to 28.8X, respectively. We analyzed the subsequence importance of the input multivariate time series. The results show that subsequences near the center of the pre-miRNA sequences are more important. This aligns with the findings from previous work. This study demonstrates that time series analysis provides a powerful alternative to conventional modeling in the context of RNA processing. This framework may be extended to other RNA-processing tasks. Notably, the encoding of RNA sequence into time series enables us to utilize any well-established tools from the time series community.

Chapter 4

Scaling with Multiple Scaling Factors in Time Series Searching

Time series data are ubiquitous across many different fields. Many data mining tasks, such as classification, clustering, and motif finding, have been defined for time series data. They utilize similarity search as a core subroutine, making it crucial to design similarity measures that align with our intuitions. To facilitate efficient computation, speedup techniques are essential. Dynamic Time Warping (DTW) is arguably the most prevailing distance measure for time series data. However, studies have shown that for certain data, another distance measure, namely Uniform Scaling (US), is equally crucial as DTW. DTW handles the local distortion, while US handles the global scaling. In addition, studies have demonstrated that combining DTW and US is necessary to obtain meaningful results in some cases. Surprisingly, all existing studies employ only a single scaling factor for the entire time series. A time series could consist of phases. Since each phase of a time series expresses at its own rate, using a single scaling factor is insufficient when comparing two time series that share similar phases but differ in their expression rates. We introduce the first framework that accounts for multiple scaling factors, Piecewise Scaling Distance (PSD). PSD employs other existing distance measures as subroutines. Because the naive implementation of PSD is slow, we propose a constrained version of PSD that enforces constraints based on the allowed segment lengths derived from the given scaling factor bound. It also prevents pathological results. In addition, two other speedup techniques have been proposed, which achieve 10.10X to 191.46X speedup. We also demonstrate the usage of a lower bound when DTW is used as the subroutine of PSD. Moreover, we show that the segmentation results returned by PSD can improve the accuracy of other distance measures.

4.1 Introduction

To study the mechanism of a process, we take measurements. Measurements are usually taken continuously by the sensors. Measurements of processes always yield continuous values at discrete timestamps. They are time series data. For example, smartphones collect users' GPS data. ECG monitors measure patients' heart rate. The continuous measurements compose a time series. It is not hard to see why time series data are ubiquitous across many different fields. In GPS data, each time series data point consists of the user's latitude and longitude information. They are multivariate time series. In ECG data, each data point represents the amplitude of the patient's cardiac electrical activity. They are univariate time series In this study, we focus on univariate time series.

Many data mining tasks can be defined on time series data. For example, given a time series database, we can perform clustering based on the pairwise similarity of the time series instances. A classifier can be trained when categorical labels are available. Alternatively, given a long time series, for motif finding, we identify recurring patterns. In contrast, for anomaly detection, we identify abnormal subsequences. Almost all time series data mining tasks can be reduced to arguing the similarity between two time series. A good distance measure, also known as a similarity measure, can determine the success or failure of the algorithms built on it. The choice of an appropriate distance measure is particularly evident in classification. Studies show that simple nearest-neighbor classification (1-NN) is difficult to beat and can compete with more complex methods [7].

A time series is treated as a whole rather than as a collection of individual values. The relationships between values are important. They constitute trends and shapes. Hence, similarity search in time series data is approximate-based rather than exact match-based [22]. Besides, different invariances should be allowed during the comparison.

Dynamic Time Warping (DTW) is one of, if not the most common, similarity measures. DTW provides invariance to time distortion by aligning and measuring the similarity between two series that may be misaligned in time. However, it assumes that the time series are expressed on a similar global expression rate. This assumption limits its performance when comparing two time series expressed at different global expression rates. We often see this behavior in domains such as speech recognition, motion analysis, patient biomedical signals, and sensor data in the manufacturing industry.

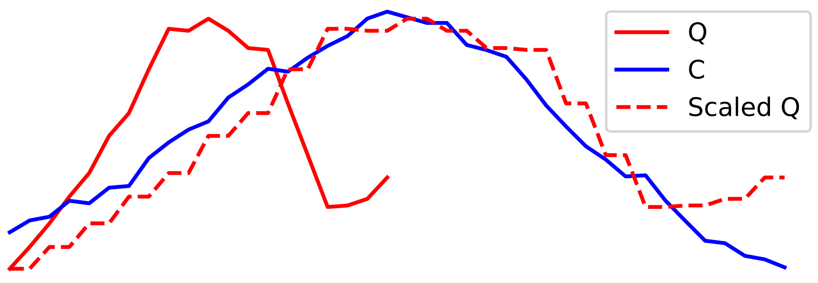


Figure 4.1: Applying nearest neighbor interpolation on Q, which result in Scaled Q, that can better reflect its similarity with C.

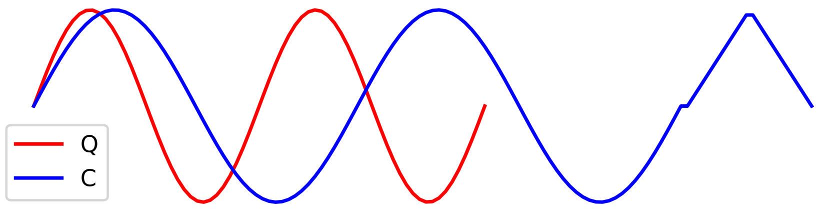


Figure 4.2: Q and C are in different rates. A stretching version of Q is similar to a prefix of C, but not the whole C.

Uniform Scaling (US) can achieve global scaling invariance by scaling the two time series to the same length via interpolation, such as nearest-neighbor interpolation, before comparison, as shown in Figures 4.1. It is reported that in some domains, such as gestures [51, 37] and music performance [14], the scaling is about 10-15% (i.e., scaling factors: 1.1 to 1.15). The scaling factors are relatively small, since the nature of the music and the gait will change with significant scaling factors. However, in some other domains, we may encounter larger scaling factors. In bioinformatics, gene expression time series data could differ by a factor of 1.41 [36, 9]. In Figure 4.2, Q and a prefix of C are similar, but at different rates. In searching, we typically have a query Q and a longer candidate C. We seek a

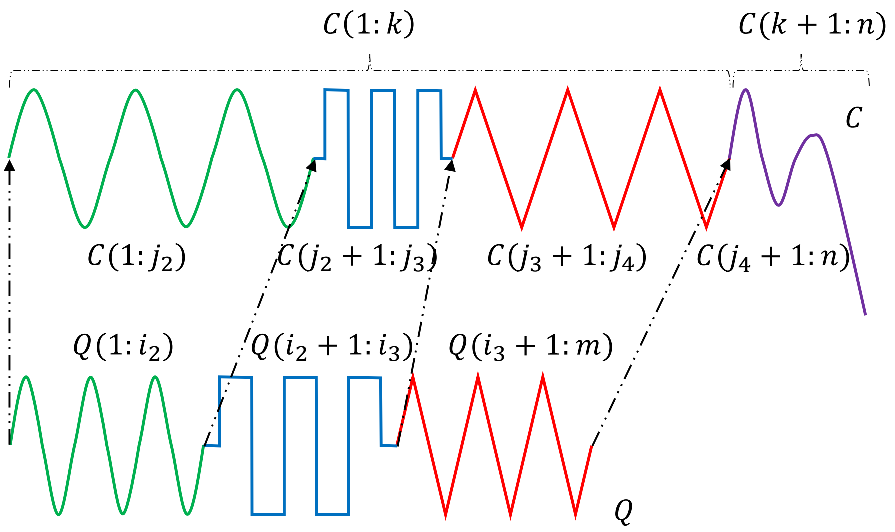


Figure 4.3: Intuition of piecewise scaling (PS).

prefix of C that is close to Q. For better comparison, we need to eliminate the scaling effect. These observations demonstrate the necessity of uniform scaling.

DTW and US are used to achieve different kinds of invariance. DTW handles local distortion, while US handles global scaling. Furthermore, some studies show that the combination of US and DTW, namely USDTW, better reflects similarity [21, 55, 56]. US is first applied to transform the two time series into the same length to eliminate the effect resulting from the different rates. Then, DTW, rather than ED, is applied to address local misalignment. USDTW is computationally more expensive than DTW because it involves the calculation of the DTW between Q and different lengths of each prefix of C. The different lengths of the prefixes correspond to different scaling factors.

It is not uncommon for the data sampling strategy to change over time [2]. There are different phases, each with its own rate. To achieve invariance for this kind of scaling effect resulting from multiple rates, rather than using a single scaling factor, it is beneficial to identify these different phases and use the appropriate scaling factors for these segments, also known as pieces. We refer to this as piecewise scaling (PS). Figure 4.3 shows the intuition of PS. The prefix of C (i.e., $C(1 : k)$) and Q share the same set of segments, but each has a different scaling. Multiple scaling factors must be used. It motivates us to design a new distance measure or framework that considers applying a scaling factor on each of the phases as defined by dashed lines in Figure 4.3, during the comparison of two time series.

Our contributions are as follows:

- We propose the first framework to achieve piecewise scaling (PS) invariance. In particular, we focus on two instantiations of PSD, namely PSED (i.e., ED with PS invariance) and PSDTW (i.e., DTW with PS invariance).

- We design a dynamic programming method to compute PSD.

- We propose a constrained version of PSD (cPSD) based on the allowed segment lengths. Besides, two other speedup techniques have been proposed. For a particular instantiation of PSD, PDTW, we demonstrate the usage of a lower bound to further speed it up.

- We demonstrate that the segmentation results returned by PSD can improve the accuracy of other distance measures.

The rest of this paper is structured as follows. We present related work in Section 4.2 and preliminaries in Section 4.3. Section 4.4 introduces our new distance measure framework, its constrained version, and speedup techniques. It is experimentally demonstrated in Section 4.5 for the problem of querying. In Section 4.6, we conclude this study with some future work.

4.2 Related Work

This study focuses on distance measures of time series. For the overall review of time series, we direct the readers to [22, 19] for a more comprehensive understanding of this field.

For many tasks, having appropriate distance measures that align with our intuition for the domains we work with is essential. One well-known distance measure is Dynamic Time Warping (DTW). It is initially designed for speech analysis [53]. However, DTW is computationally expensive. Lower bounds are used to speed up time series similarity search by admissibly pruning the unpromising candidates. One of the popular exact lower bounds of DTW is $\mathrm{LB}_{\mathrm{Keogh}}$. [49] improves the scalability of DTW by introducing a subsequence search suite of their four novel ideas, namely the UCR suite. For an overall review of lower bounds, we refer readers to [60, 57]. There is an approximate algorithm that approximates DTW with high accuracy while drastically cutting down the time and space requirements [54].

While ED is sensitive to distortions in the time axis, uniform scaling (US) has been shown to be a critical invariance in domains such as motion capture. [30] demonstrated that DTW is insufficient for handling global scaling effects, and that identifying DTW is not the solution to achieve this kind of invariance. There is a need for US. [64] extends the importance of uniform scaling to motif discovery. The authors show that meaningful motifs often suffer from a global scaling effect, causing standard motif finding algorithms to miss them completely.

To the best of our knowledge, three studies analyze the combination of US and DTW, namely USDTW. It was first proposed by [21]. It extended LB_{Keogh} to bound the USDTW. However, the extended LB_{Keogh} is still too loose with invariance to large amounts of uniform scaling. [55] and its follow-up study [56] proposed a new lower bound, namely LB_{Shen} [1] , which has been shown to be tighter than LB_{Keogh} on USDTW.

To our surprise, despite a fruitful discussion of DTW, US, and USDTW, no study has proposed a distance measure capable of handling scaling effects across multiple scaling factors. This is precisely what we will address in this study.

4.3 Preliminaries

We refer to time as the contextual attribute because it provides the context for the measurements to be made. We refer to the measurements as the behavioral attributes. Time series are multivariate when more than one behavioral attribute is present. Otherwise, it is called univariate. We focus on the univariate case.

Definition 1 (Time Series)**.** A time series $T = t_1, t_2, ..., t_n$ is a sequence of real-valued numbers with length $= n$.

When two time series are involved in the discussion, we denote them as Q (Query) and C (Candidate), with lengths m and n, respectively. Since Q is the query sequence, it is not longer than C (i.e., $m \leq n$). The requirement of "$m \leq n$" is a natural setting. In "Query by Content", a user is going to search for a candidate in the database from a user-input query in which the query may only contain partial

[1]It is denoted as LB_{New} in the original study. We rename it to prevent any confusion.

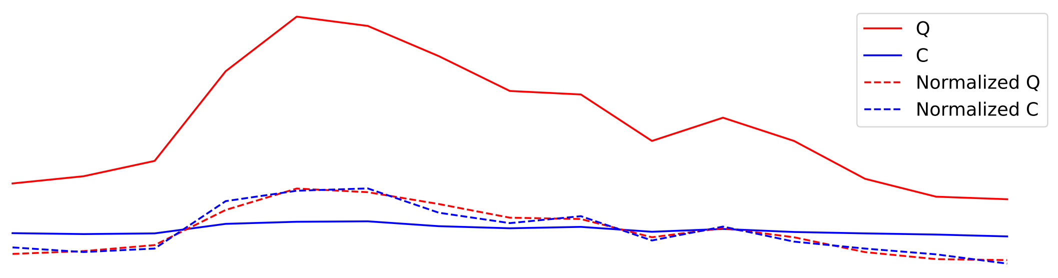


Figure 4.4: Z-normalization. Resulting time series have mean = 0 and std = 1.

information of the target candidate. For example, a user often wants to find a song or tune that is lingering in their head by humming a part but not whole of the tune [68]. We are also interested in a segment or subsequence of a time series.

Definition 2 (Subsequence)**.** A subsequence $T(i:j)$ of a time series T is a shorter time series that starts from position i and ends at position j with length $= j-i+1$. Both ends are inclusive. Formally, $T(i:j) = t_i, t_{i+1}, ..., t_j$, $1 \leq i \leq j \leq n$.

We call $T(1:j)$ the prefix of T of length j.

Before comparison, we need to standardize or normalize them. A common way is Z-Normalization, which is $T = (T - \text{mean}(T))/\,\text{std}(T)$, as shown in Figure 4.4. The most fundamental distance measure is the Euclidean Distance (ED).

4.3.1 Euclidean Distance (ED)

Given two points on a plane, it is intuitive to define the distance between them as the length of the line segment between them. This idea extends to the case of n-dimensions in time series with length n.

Definition 3 (Euclidean Distance (ED))**.** Given two series Q and C both with length n, the Euclidean Distance between them is defined as:

$$\mathrm{ED}(Q, C) = \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2} \tag{4.1}$$

A square root is usually involved in the computation of distance measures. It is a monotonic function. Since it does not change the relative ranking of the results, we can omit the square root operation for simplicity and optimization. ED aligns the entries between two series in a one-to-one manner. Two similar series will have a large distance under ED if they are not aligned well in the time dimension. ED cannot handle local distortion along the time axis because warping in the alignment is not allowed. A standard distance measure that provides warping invariance is called Dynamic Time Warping (DTW).

4.3.2 Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) is an algorithm that measures similarity between two time series while accounting for local distortions. DTW aligns the two series by nonlinearly warping the time axis to minimize the final cumulative distance.

Given two time series, Q and C, we first construct an m by n distance matrix M. The origin (1, 1) is set at the bottom-left element of M. The (i, j) element of M contains the distance $\mathrm{d}(q_i, c_j)$ between points q_i and c_j. This local distance $\mathrm{d}(\cdot, \cdot)$ is usually calculated by $(q_i - c_j)^2$. Each (i, j) element refers to an alignment or mapping of the two points. A contiguous set of such elements forms a warping path W. W represents the non-linear alignment of Q and C. $W = w_1, w_2, ..., w_K$

in which $w_k = (i, j)_k$ represents the mapping between q_i in Q and c_j in C, where $\max(m, n) \le K \le m+n-1$. "$\max(m, n) \le K$" because the alignment of two series must include every point in both Q and C. "$K \le m + n - 1$" because the longest warping path is either the "concatenation of the bottom row and the rightmost column" (with the bottom-right cell being overlapped) or the "concatenation of the leftmost column and the top row" (with the top-left cell being overlapped).

The warping path W is typically subject to the following three constraints.

- Boundary conditions: $w_1 = (1, 1)$ and $w_K = (m, n)$. The first (last) point of Q must map to that of C.
- Continuity: Given $w_k = (i, j)$ and $w_{k-1} = (i', j')$, $i - i' \le 1$ and $j - j' \le 1$. An entry in the warping path W is adjacent to its one-step previous entry.
- Monotonicity: Given $w_k = (i, j)$ and $w_{k-1} = (i', j')$, $i - i' \ge 0$ and $j - j' \ge 0$. The warping path W does not go back.

We denote W^* as a set of all allowed possible paths.

Definition 4 (Dynamic Time Warping (DTW) [53])**.** DTW returns the minimum warping cost:

$$\text{DTW}(Q, C) = \min_{W \in W^*} \sum_{k=1}^{|W|} w_k \tag{4.2}$$

To find the minimum warping cost and its corresponding warping path, we can use dynamic programming (DP) to evaluate the following recurrence.

$$D(i,j) = \mathrm{d}(q_i, c_j) + \min \begin{cases} D(i-1, j-1), \\ D(i-1, j), \\ D(i, j-1) \end{cases} \tag{4.3}$$

It can be solved by building the accumulated cost matrix D, where the y-axis refers to Q, and the x-axis refers to C. The base cases, which are the first row and the first column, are defined as $D(1,j) = \sum_{k=1}^{j} \mathrm{d}(q_1, c_k), j \in [1,n]$ and $D(i,1) = \sum_{k=1}^{i} d(q_k, c_1), i \in [1,m]$. After we have initialized the base cases, we can fill up D starting from the bottom up. There are $m \times n$ entries in D. It takes $\mathcal{O}(mn)$ to fill it up. Once D is built, we can find the path corresponding to this minimum cost by simple backtracking from the end cell $D(m,n)$ to the origin cell $D(1,1)$.

Some constraints have been proposed to prevent pathological warping paths with the aim of accuracy and efficiency. Two of the most popular are Sakoe-Chiba Band [53] and Itakura Parallelogram [28]. They only allow the warping paths to pass through the allowed region, as shown in Figure 4.5, by setting the cells outside this region in D to have ∞ accumulated distance cost. [31] suggested that these constraints can be viewed as constraints on the warping path entry $w_k = (i,j)_k$. It represents these constraints as inequalities applying to the indices i and j locally, without depending on the main diagonal in D. In the Sakoe-Chiba Band, the constraints can be represented as $j - r \leq i \leq j + r$, where r is an integer. It is sometimes specified as a fraction (or percentage) of the longer time series length

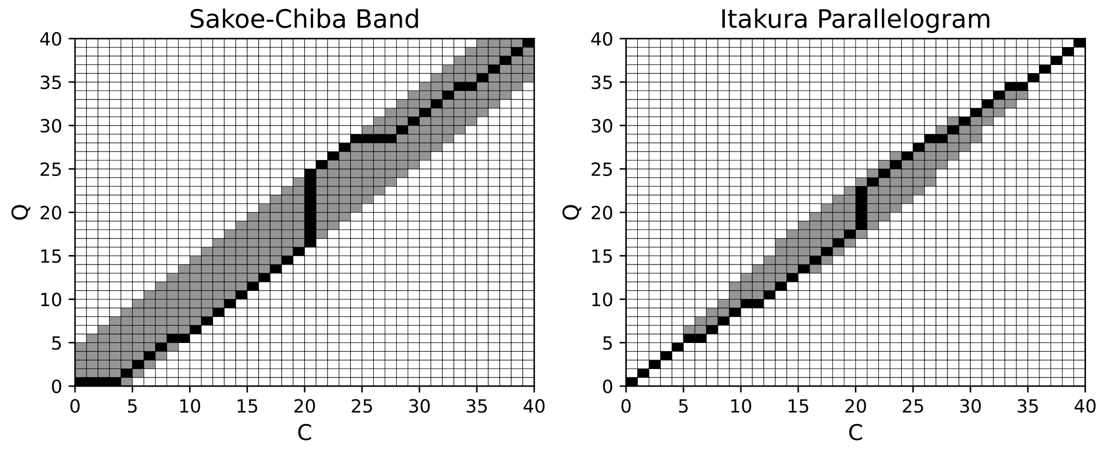


Figure 4.5: Visualization of D with local constraints. The black cells form the warping path.

to ensure length invariance. For clarity, we assume r is an integer unless specified otherwise. This means that q_i can only align with c_j if their indices differ by at most r. Since r defines the maximum allowed difference between the mapping indices, $|n - m| \leq r$, to ensure that the end points of Q and C can map. In the Itakura Parallelogram, r is a function of i rather than a constant value. ED can be seen as a special case of DTW where the warping path is fixed to be diagonal. q_i aligns to c_i for every i (i.e., $r = 0$). DTW minimizes over all possible warping paths, and the warping path of ED is one of them. Because of the band, we only need to fill up the cells within the band. The complexity is $\mathcal{O}(\#\_of\_cells\_inside)$. In the case of the Sakoe-Chiba Band, the band has a constant width $w = 2r + 1$. The complexity becomes $\mathcal{O}(w \times length\_of\_diagonal) = \mathcal{O}(rn)$. The constrained DTW is denoted as DTW$_r$.

4.3.3 Uniform Scaling (US)

In US, we compare the whole sequence of Q to a prefix of C, as shown in Figure 4.2. The two compared sequences are scaled to the same length via interpolation before ED is applied. A common interpolation method is nearest neighbor interpolation.

Definition 5 (Nearest Neighbor Interpolation)**.** Given a time series T of length n and an integer L , Nearest Neighbor Interpolation scales T into T^L as follows:

$$T_j^L = T_{\lceil n(j/L) \rceil)} \quad where\ 1 \leq j \leq L \tag{4.4}$$

We can scale up or down a given series using Equation 4.4, as shown in Example 1.

Example 1. Given a series $T = 1, 2, \cdots, 6$ with length $n = 6$. Let $T(1:4)$ be the prefix of T of length $k = 4$ (i.e., $T(1:4) = 1, 2, 3, 4$). Given an integer $L = 8$, we compute $T(1:4)^8$ as follows.

$$\begin{aligned} T(1:4)^8 &= T_{\lceil 4(1/8) \rceil}, T_{\lceil 4(2/8) \rceil}, T_{\lceil 4(3/8) \rceil}, \ldots, T_{\lceil 4(8/8) \rceil} \\ &= T_1, T_1, T_2, \ldots, T_4 \\ &= 1, 1, 2, \ldots, 4. \end{aligned}$$

When $L > k$, T is said to be stretched. When $L < k$, T is said to be shrunk.

Definition 6 (Uniform Scaling (US) [30])**.** Given two series Q and C, of length m and n respectively, and a scaling factor bound l, where $l \geq 1$. Let $C(1:k)$ be the prefix of C, where $\lceil m/l \rceil \leq k \leq \min(\lfloor lm \rfloor, n)$, and $C(1:k)^L$ be a rescaled version of $C(1:k)$ with length L, where $L = \min(\lfloor lm \rfloor, n)$. L is called the alignment factor. $\min(\lfloor lm \rfloor, n)$ is the largest alignment factor.

$$\mathrm{US}(Q, C, l, L) = \min_{k=\lceil m/l \rceil}^{\min(\lfloor lm \rfloor, n)} \mathrm{ED}(Q^L, C(1:k)^L) \tag{4.5}$$

L is set as the largest alignment factor [56] to ensure all the points in Q and $C(1:k)$ are preserved during interpolation because of up-sampling, and the scaled version of all the prefixes is going to have the same length (i.e., L), for fair comparison, since comparison between two longer time series generally results in a larger distance measure. Through Equation 4.5, we find the minimum value and the corresponding argument (i.e., k) by checking the minimum value of the ED function from $\lceil m/l \rceil$ to $\min(\lfloor lm \rfloor, n)$. The scaling factor is defined by the argument minimum value of k. The smaller k is, the more we need to "stretch" C for Q to compare with $C(1:k)$, which is m/k times.

Consider a time series database D comprising a set of candidate instances, and a single query series Q. The search task aims to retrieve the most similar instance (or top-k instances) from D to Q. We maintain Q as a fixed reference and extract only the prefixes from each instance in D for comparison. Furthermore, to simplify notation, we apply scaling exclusively to the instances in D, leaving Q unscaled.

4.3.4 Uniform Scaling & Dynamic Time Warping (USDTW)

Uniform Scaling & Dynamic Time Warping (USDTW) measures the similarity by applying scaling with an appropriate scaling factor on C and then applying DTW. The tail part of C can be ignored, as shown in Figure 4.2.

Definition 7 (Uniform Scaling & Dynamic Time Warping (USDTW) [56])**.** With the same notations defined in Definition 6,

$$\begin{aligned}&\mathrm{USDTW}_r(Q, C, l, L) = \\ &\qquad \min_{k=\lceil m/l \rceil}^{\min(\lfloor lm \rfloor, n)} \mathrm{DTW}_r(Q^L, C(1:k)^L)\end{aligned} \tag{4.6}$$

where r is the DTW constraint parameter.

We replace the ED function in Equation 4.5 by DTW function to form Equation 4.6.

As mentioned in Section 4.1, there may exist more than one scaling factor. Hence, both the existing distance measures, US and USDTW, which are designed to handle only one scaling factor, are insufficient. This motivates us to design a new framework of distance measures.

4.3.5 Lower Bounds for DTW and USDTW

We first introduce the concept of lower bounds and explain how they can benefit search. DTW is computationally more expensive than ED. They compute an accumulated cost matrix of size $m \times n$, where m and n denote the lengths of the two time series. It results in quadratic complexity. This complexity poses a challenge for similarity search. The most common approach is to compute a lower bound of the real value, which is computationally cheap and tight. We can use this lower bound to filter out unpromising candidates and perform the expensive distance computation only on the small set of promising candidates. In searching, we would keep track of the best_so_far distance bsf between the query Q and the testing candidates. When testing a new candidate C, we first compute the $\mathrm{LB}(Q, C)$. We only compute the actual DTW when $\mathrm{LB}(Q, C) \leq \mathrm{bsf}$.

We then introduce some common lower bounds.

Kim Lower Bound [33]: $\mathrm{LB}_{\mathrm{Kim}}$ is a simple and fast lower bound of DTW. The complexity is $O(1)$. It uses the four features in Q and C. We denote t_{-1} as the last point and $t_{\max}$ ($t_{\min}$) as the maximum (minimum) point in time series T.

$$\mathrm{LB}_{\mathrm{Kim}} = \max \begin{cases} \mathrm{d}(q_1, c_1) \\ \mathrm{d}(q_{-1}, c_{-1}) \\ \mathrm{d}(q_{\max}, c_{\max}) \\ \mathrm{d}(q_{\min}, c_{\min)} \end{cases} \tag{4.7}$$

$\mathrm{d}(\cdot, \cdot)$ refers to the local distance used in the point alignment. The first two lines come from the boundary condition in DTW. The alignment between the first pair of points and the last pair of points must contribute to the accumulated sum of DTW. Each point in Q must align with some point in C, and vice versa. Each alignment contributes a local distance to the final sum. The minimum possible local distance for the alignment of $q_{\max}$ would be the one aligned with $c_{\max}$. The same applies to the last line in the equation.

There is a simplification of $\mathrm{LB}_{\mathrm{Kim}}$. Only the first and last pair are used in the lower bound computation. In the normalized time series, the third and fourth rows in Equation 4.7 should have small values [49]. Ignoring them can improve the complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. The simplified lower bound is $\mathrm{LB}_{\mathrm{KimFL}} = \mathrm{d}(q_1, c_1) + \mathrm{d}(q_{-1}, c_{-1})$.

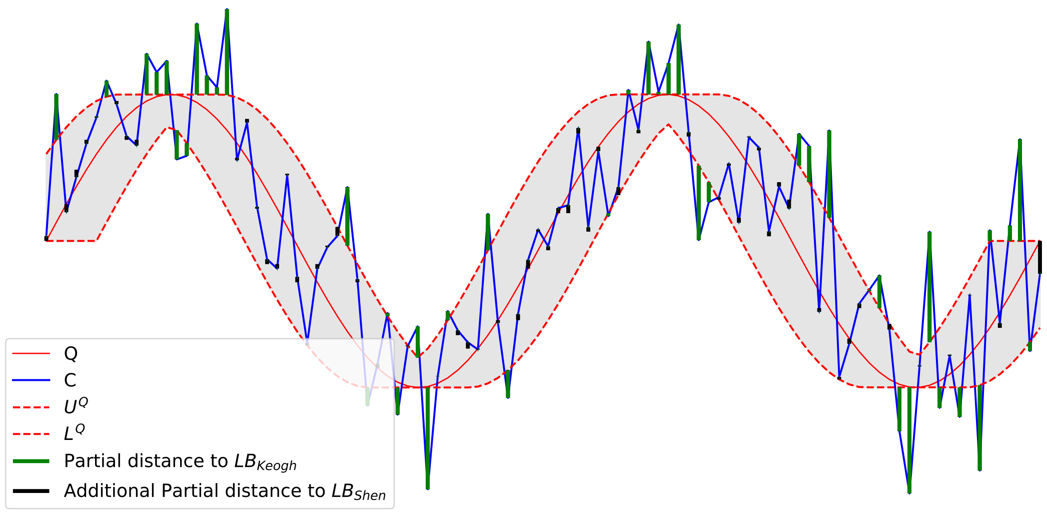


Figure 4.6: Visualization of LB_{Keogh} and LB_{Shen}

Keogh Lower Bound [31]: LB_{Keogh} builds the upper U and lower envelopes L of one of the time series out of the two compared sequences. Usually, the envelopes are constructed for Q instead of C as we will compare one Q against many C's. Otherwise, we need to build the envelopes for each candidate instead [49].

To the best of our knowledge, they are the first to interpret the constraint as a restriction on the indices of the warping path $w_k = (i, j)_k$ such that $j - r \leq i \leq j + r$, where r defines the allowable deviation of alignment between q_i and c_j. For ease of exposition, we focus on the most used constraint in the literature, which is the Sakoe-Chiba Band [59]. Two sequences are constructed for Q, namely the upper U^Q and lower envelopes L^Q of Q as shown in Figure 4.6. For each q_i, we would assign a window of q_i based on its index i as follows.

$$
\begin{aligned}
U_i^Q &= \max(q_{\max(1,i-r)} : q_{\min(i+r,m)}) \\
L_i^Q &= \min(q_{\max(1,i-r)} : q_{\min(i+r,m)})
\end{aligned}
\tag{4.8}
$$

$\max(1,\cdot)$ and $\min(\cdot,m)$ are used for handling the boundary cases. U^Q and L^Q together form a bounding envelope that encloses the original Q, it is the grey region in the figure. For each c_j, either (c_j, U_j^Q) or (c_j, L_j^Q) corresponds to the possible alignment that contributes the minimum distances if c_j falls outside the envelope. Herein, the lower bound is the sum of these distances, as shown in the following equation.

$$
\mathrm{LB}_{\mathrm{Keogh}}(Q,C) = \sum_{j=1}^{n} \begin{cases} d(c_j, U_j^Q) & \text{if } c_j > U_j^Q \\ d(c_j, L_j^Q) & \text{if } c_j < L_j^Q \\ 0 & \text{otherwise} \end{cases}
\tag{4.9}
$$

Visually, these distances are the distances between c_j outside the envelope and the vertically corresponding points on the envelope. The distances are the green bars in the figure. Equation 4.9 returns the sum of the green bars.

Shen Lower Bound [55, 56]: It leverages the boundary and continuity conditions to create a lower bound of DTW. It can be used to lower-bound the USDTW with slight modification. The intuition is to find the minimum possible alignment of each c_j with points in Q, subject to the local constraint r from DTW and the global constraint l from US.

We will first introduce $\mathrm{LB}_{\mathrm{Shen}}$ for DTW. Given the candidate sequence C with length n, for each c_j, we create its possible reach $\mathbb{q}_j$ in Q under DTW as in Equation 4.10. The elements $\mathbb{q}_j$ form an indexed collection $\mathbb{Q} = (\mathbb{q}_1, \mathbb{q}_2, \ldots, \mathbb{q}_{\min(\lfloor lm \rfloor, n)})$.

$$\mathbb{q}_j = (q_{\max(1,j-r)}, \ldots, q_{\min(j+r,m)}) \tag{4.10}$$

We define $\delta(x, Y) = \min_{y \in Y} d(x, y)$. The possible minimum cost contributed by the alignment of c_j and some point in Q to the accumulated sum would be $\delta(c_j, \mathbb{q}_j)$. The lower bound LB_{Shen} is defined as:

$$\text{LB}_{\text{Shen}}(Q, C) = \text{d}(c_1, q_1) + \sum_{j=2}^{n-1} \delta(c_j, \mathbb{q}_j) + \text{d}(c_n, q_m) \tag{4.11}$$

We direct the reader to [55] for the formal proof, while an intuition of the proof is presented here. There are three items on the right-hand side of Equation 4.11. The continuity requirement ensures that each c_i is involved in at least one alignment. The first and the last items are from the boundary condition. The middle item returns the possible minimum distances contributed by each c_j's, where $2 \leq j \leq n-1$. It is obvious that LB_{Shen} in Equation 4.11 is tighter when we use the first pair of points and the last pair of points instead of doing the middle summation all from $j = 1$ to n. It is because the distance contributed by the first pair (last pair) must be greater than $\delta(c_1, \mathbb{q}_1)$ ($\delta(c_n, \mathbb{q}_n)$).

It is proven that it is tighter than LB_{Keogh} [55]. It is shown in Figure 4.6 visually. The black bars refer to the additional lower bound distance sum on top of LB_{Keogh}. We will give an intuitive proof here. Both LB_{Keogh} and LB_{Shen} compute the lower bounds by summing the local distance resulting from the alignment of each c_j with some points in Q, which is guaranteed to be not greater than the partial distance contributed by the real alignment, which we only know until we compute the exact distance measure. In LB_{Keogh}, if this c_j falls outside the envelope, this local distance is the vertical distance between c_j and the envelope. If c_j falls inside, this local distance would be 0. For those points outside the envelope, the local distances for c_j of $\mathrm{LB}_{\mathrm{Keogh}}$ and $\mathrm{LB}_{\mathrm{Shen}}$ are the same. But $\mathrm{LB}_{\mathrm{Shen}}$ aims to return the minimum possible partial distance for each c_j, even within the envelope. Hence, $\mathrm{LB}_{\mathrm{Keogh}} \leq \mathrm{LB}_{\mathrm{Shen}}$.

In order to compute Equation 4.11 efficiently, we first sort sequences $\mathbb{q}_j$ and the resulting sorted sequences are denoted as $\tilde{\mathbb{q}}_j$. The sorted sequences allow us to do a binary search when we are calculating $\delta(c_j, \tilde{\mathbb{q}}_j)$ in contrast to $\delta(c_j, \mathbb{q}_j)$. The sorting only needs to be done once because we are testing the same Q with different candidates.

It can be extended to the USDTW case [55, 56]. The possible reach now is not only defined by r, but also by the scaling factor bound l.

$$\mathbb{q}_j = (q_{\max(1,\lceil j/l \rceil - r)}, \ldots, q_{\min(\lfloor jl \rfloor + r, m)}) \tag{4.12}$$

[55] proves the following theorem to allow us to consider the lower bound between each prefix of C and Q without the scaling up of each prefix of C (i.e., $C(1:k)^L$) and the scaling up of Q (i.e., Q^L).

Theorem 8. *For any* $\lceil m/l \rceil \leq k \leq \min(\lfloor lm \rfloor, n)$, $\mathrm{DTW}_r(Q^{\min(\lfloor lm \rfloor, n)}, C(1:k)^{\min(\lfloor lm \rfloor, n)})$ *is always lower bounded by* $\sum_{j=1}^{k} \delta(c_j, \mathbb{q}_j)$.

Recall that USDTW calculates DTW distances between each rescaled prefix of C to the rescaled Q, and outputs the minimum DTW distance, as in Equation 4.6. The incremental nature of the lower bound frees us to calculate the lower bound of each DTW distance from scratch. This theorem allows us to first calculate the lower bound of the shortest prefix $C(1:\lceil m/l \rceil)$ of C and Q, and then incrementally calculate the lower bound of the longer prefix with length from

$\lceil m/l \rceil + 1$ to $\min(\lfloor lm \rfloor, n)$ by adding on each $\delta(c_j, \mathbb{q}_j)$. To note, it also means that if $\text{LB}_{\text{Shen}}(Q, C(1:k))$ is larger than a value, namely bsf, $\text{LB}_{\text{Shen}}(Q, C(1:k'))$, where $k' > k$, would also be larger than bsf. To note, we can tighten LB_{Shen} by using $\text{d}(c_1, q_1)$ instead of $\delta(c_1, \mathbb{q}_1)$.

The above analysis can also apply to the case of US distance. We only need to define the corresponding reaches as follows:

$$\mathbb{q}_j = \left(q_{\max(1,\lceil j/l \rceil)}, \ldots, q_{\min(\lfloor jl \rfloor, m)}\right) \tag{4.13}$$

4.4 Piecewise Scaling Distance

The motivation stems from the limitations of US and USDTW. They assume that the relationship between Q and C is governed by only a single, global scaling factor. This assumption fails when applied to multi-rate data, where different phases of the time series express at different rates.

Note that existing studies [30, 49] can find all scaling factors, defined by the chosen prefix of C, ranging from $\lceil m/l \rceil$ to $min(\lfloor lm \rfloor, n)$. However, they can use only one of them in the scaling. Consider the following illustrative example in ASCII text [49], where character repetition represents the duration of spoken phonemes, and space indicates a pause:

- “time   series 20 25” and “time $\underline{\text{series}\quad}$ 20 25”. Here, the Hamming distance (the discrete analogue of ED) fails due to misalignment in the underlined locations, but the string edit distance (the discrete analogue of DTW) can resolve it.

- "time <u>sseerriiss 222000222555</u>".This sequence exhibits three distinct phases with scaling factors of 1, 2, and 3, respectively. The corresponding invariance cannot be achieved by DTW or US (which is restricted to a single global scalar). US would enforce a single compromised global scale instead.

In Query-by-Content scenarios, such as query-by-humming or gesture retrieval, the query is generated by a human. Humans do not maintain a consistent rate for each phase. For example, humans rush through a familiar sequence but slow down for a new or complex sequence. It is commonly observed in a piece of music performed by a beginner. The piece's tempo is not uniform.

We introduce a novel distance measure framework, termed Piecewise Scaling Distance (PSD). It addresses the local scaling effect within each phase by employing a scaling factor to each phase, instead of using a single scaling factor for the whole time series. It releases the basic constraint or assumption made in US and USDTW. PSD employs an existing distance metric, such as ED or DTW, to quantify the similarity of aligned segment pairs. While the PSD framework is agnostic to the underlying metric, this study focuses on its two fundamental instantiations:

- PSED, which employs ED to compute the similarity of aligned segment pairs.
- PSDTW, which employs DTW to compute the similarity of aligned segment pairs.

In the formulation that follows, we utilize PSDTW as the running example. PSDTW generalizes PSED. The PSED formulation can be trivially derived from it.

4.4.1 Piecewise Scaling & Dynamic Time Warping (PS-DTW)

Problem formulation: To simplify the discussion, we focus on the comparison between Q and the entire sequence C, rather than a prefix of C. Note that this formulation can be generalized to prefix matching as in Definition 6 and Definition 7.

Given two sequences Q and C, where Q is not longer than C, $|Q| = m \leq |C| = n$, and the number of segments or pieces P allowed, our goal is to segmentalize both Q and C into P contiguous segments automatically in a way that minimize the total sum of DTW distance of aligned segment pairs with interpolation.

Definition 9 (Piecewise Scaling & Dynamic Time Warping (PSDTW))**.** With the same notations defined in Definition 7,

$$\begin{aligned}&\mathrm{PSDTW}_r(Q, C, l, L, P) = \\ &\min_{\substack{i_1<i_2<\cdots<i_{P+1}\\ j_1<j_2<\cdots<j_{P+1}}} \sum_{p=1}^{P} \mathrm{DTW}_r(Q(i_p+1 : i_{p+1})^L, \\ &\qquad\qquad C(j_p+1 : j_{p+1})^L)\end{aligned} \tag{4.14}$$

, where $i_1 = 0$, $i_{P+1} = m$, $j_1 = 0$, $j_{P+1} = n$ and the setting of L will be discussed later.

Essentially, L needs to be at least the length of all segments to preserve all the points by up-sampling.

We refer to Figure 4.3 to clarify Equation 4.14. Given two sequences Q and C, with the aid of different colors and the dashed lines, we observe that Q consists of

Algorithm 1 Naive PSDTW

```
Input: Query series $Q$, Candidate series $C$, DTW constraint parameter $r$ (in fraction), Number of pieces $P$, Scaling parameter $L$
Output: Distance Matrix $D$ of size $(m+1) \times (n+1) \times (P+1)$
1: Initialize $D$ with $\infty$
2: $D[0,0,0] \leftarrow 0$
3: for $p \leftarrow 1$ to $P$ do
4:     for $i \leftarrow 1$ to $m$ do
5:         for $j \leftarrow 1$ to $n$ do
6:             $D[i,j,p] \leftarrow \min_{i'<i,j'<j} \{D[i',j',p-1]$
                   $+ \mathrm{DTW}_r(Q(i'+1:i)^L, C(j'+1:j)^L)\}$
7: return $D[m,n,P]$
```

three segments while C consists of four segments, with the first three segments of C similar to those of Q. They form three segment pairs. The scaling factor used in each segment pair is determined from the length of the two subsequences involved, i.e., $(i_{p+1} - i_p)/(j_{p+1} - j_p)$. These three parts have different scaling factors. For example, the first part in C is the stretched version of that in Q. The second part in C is the compressed version of that in Q.

Equation 4.14 can be formulated in a recurrence relation as follows. Let $D[i,j,p]$ be the minimum cost to align the first i points in Q (i.e, $Q(1:i)$) with the first j points in C (i.e., $C(1:j)$) using exactly p segments:

$$D[i,j,p] = \min_{\substack{i'<i \\ j'<j}} \Big\{ D[i',j',p-1] + \mathrm{DTW}_r(Q(i'+1:i)^L, C(j'+1:j)^L) \Big\} \tag{4.15}$$

Naive PSDTW:

Algorithm 2 Line 6 in Algorithm 1

```
1: for $i' \leftarrow 0$ to $i - 1$ do
2:     for $j' \leftarrow 0$ to $j - 1$ do
3:         $dist_{\text{prev}} \leftarrow D[i', j', p - 1]$
4:         $dist_{\text{seg}} \leftarrow \text{DTW}_r(Q(i' + 1 : i)^L, C(j' + 1 : j)^L)$
5:         $D[i, j, p] \leftarrow \min(D[i, j, p],\ dist_{\text{prev}} + dist_{\text{seg}})$
```

Our goal is $D[m, n, P]$. Equation 4.15 can be solved exactly by dynamic programming (DP). The base case is $D[0, 0, 0] = 0$. It refers to the zero cost to align the first 0 point (i.e., the empty prefix) of Q with that of C. Other cells in D are first initialized with ∞. They are calculated using a bottom-up approach via Equation 4.15. A straightforward implementation in DP is shown in Algorithm 1. Line 6 is achieved by looping all the previous indices of the current i and j as in Algorithm 2.

We explain the lines in Algorithm 2. Line 3 retrieves the accumulated distance cost from the beginning up to the endpoints (i', j'), and saves it as $dist_{\text{prev}}$. Line 4 considers the current aligned segment pair, which consists of $Q(i' + 1 : i)$ and $C(j' + 1, j)$, and they are interpolated to the length L. The DTW distance of this pair is calculated and saved as $dist_{\text{seg}}$.

We now analyze the time complexity of Algorithm 1. There are Pmn entries in D. The min operator in line 6 takes $\mathcal{O}(mn)$. Hence, the time complexity of Algorithm 1 is $\mathcal{O}(Pm^2n^2) = \mathcal{O}(Pn^4)$, multiplied by the running time of the DTW. It is slow, which prevents us from using it in practice. To note, we use r in a fraction instead of a fixed integer here. This allows the deviation tolerance to scale adaptively with pieces of varying lengths.

Algorithm 3 Initialization of PSDTW

```
1: $L^Q_{\text{gmin}} \leftarrow \lceil (m/P)/\sqrt{l} \rceil, \quad L^Q_{\text{gmax}} \leftarrow \lfloor (m/P)\sqrt{l} \rfloor$        ▷ "g" refers to "global".
2: $L^C_{\text{gmin}} \leftarrow \lceil (n/P)/\sqrt{l} \rceil, \quad L^C_{\text{gmin}} \leftarrow \lfloor (n/P)\sqrt{l} \rfloor$
3: $L = \max(L^Q_{\text{gmax}}, L^C_{\text{gmin}})$
4: Initialize $D$ of size $(m+1) \times (n+1) \times (P+1)$ with $\infty$
5: $D[0,0,0] \leftarrow 0$
```

4.4.2 Speedup Techniques

Length constraints of the segment: A way to reduce complexity and to prevent pathological segment pairs is to limit the possible segment lengths that are considered by constraining the minimum and maximum lengths of segments. It is similar to constrained DTW, in which we limit the search space of the warping path as in Figure 4.5. The version of PSDTW that considers the segment constraint is termed constrained PSDTW (cPSDTW). It is shown in Algorithm 4. The uncolored part shows the main logic, while the colored part shows the speedup techniques, which will be explained later.

We initialize in Algorithm 3. For a given number of segments P, the expected length of each segment in Q and C would be m/P and n/P, respectively. For Q, we set the minimum possible segment length L^Q_{gmin} to be $\lceil (m/P)/\sqrt{l} \rceil$ and the maximum possible length L^Q_{gmax} to be $\lfloor (m/P)\sqrt{l} \rfloor$ in line 1 such that the scaling ratio of any two segments in Q would be bounded by l. It allows some deviation in the length of the segments from their expected length. Similarly, we compute the segment constraints for C in line 2. We set the maximum segment length be the alignment factor L in line 3. We set the base case in line 5.

We fill the table D in Algorithm 4. In line 3, given p pieces in Q, the ending index of the last piece (i.e., the p^{th} piece) will range from $(p \cdot L^Q_{\text{gmin}})$, given all the

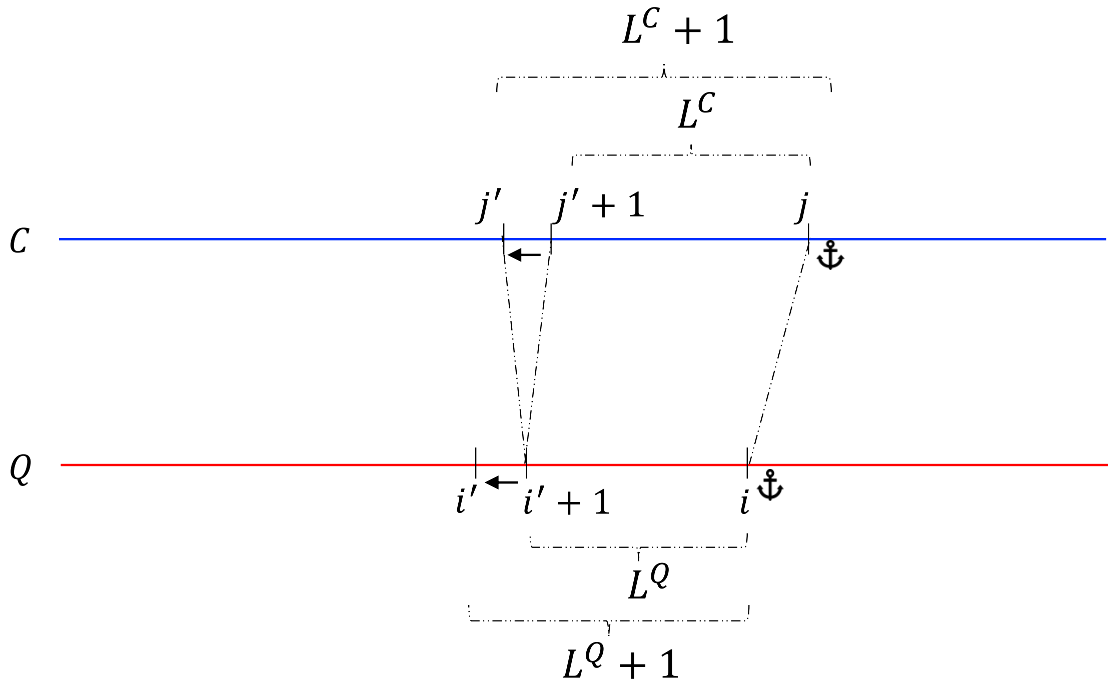


Figure 4.7: Relationship of L_Q and L_C.

p pieces are in minimum length L^Q_{gmin}, to $\min(p \cdot L^Q_{\text{gmax}}, m)$, given all the p pieces are in maximum length L^Q_{gmax}.

In line 4, we enumerate for all the allowed lengths L^Q. For the segment with length L^Q in Q, the length L^C of the corresponding aligned segment in C will have a range from $L^C_{\min}$ to $L^C_{\max}$, as defined in lines 7–8, such that the ratio of L^Q and L^C would be bounded by l. Because L^Q and L^C increase in line 4 and line 14, and the i and j are fixed by the outer loop, in line 3 and line 13, respectively, the i' and j' decrease correspondingly It is visualized in Figure 4.7. A segment in Q with ending index i and starting index $i' + 1$ would be compared to a set of segments in C, all of which have a fixed end at j and a decreasing starting index, starting at $j' + 1$. For example, the starting index of the first segment with being compared is $j' + 1$, which has length L^C, and that of the second segment is j', which has length $L^C + 1$, as in Figure 4.7.

In line 18, we terminate the current iteration if there are no previously valid segment pairs (i.e., $dist_{\text{prev}} = \infty$).

In line 22, if the accumulated cost $dist_{\text{prev}}$ exceeds the current best so far, we stop the current iteration as the resulting $(dist_{\text{prev}} + dist_{\text{seg}})$ is guaranteed to be greater than the current best so far, which is stored in $D[i, j, p]$.

To be consistent with the result of using the lower bound speedup technique, which will be introduced later, we compute the DTW in the reverse manner in line 33. Due to the nearest neighbor interpolation, $\text{DTW}_r(Q'^L, C'^L)$ may not equal to $\text{DTW}_r(\text{rev}(Q')^L, \text{rev}(C')^L)$.

In line 35, we also save the pairs (i', j') that serve as cutting points between segments to obtain the segmentation result.

Parallel computing: We observe that the recurrence relation for the state $D[i, j, p]$ at stage p depends exclusively on the states computed at stage $p - 1$, as shown in Equation 4.15. There are no stage dependencies over indices i and j. It allows us to parallelize the loops over indices i and j on lines 3 and 13. In the following experimental section, we distribute the i-loop iterations (i.e., line 3 in Algorithm 4, which is colored in blue) across available threads only because there are already sufficient iterations from the i-loop to fill the available threads. There are $\Big(\min(p \cdot L^Q_{\text{gmax}}, m) - (p \cdot L^Q_{\text{gmin}}) + 1\Big)$ iterations from the i-loop. The parallel execution of the i-loop is implemented by `prange` in `Numba` in Python.

Early Abandoning in nearest neighbor search: To accelerate the nearest neighbor search (or top-k search) for a query Q on a candidate set, we employ an early abandoning strategy. It prunes the search branch within the PSD computation of a specific candidate C as soon as the result corresponding to this search branch is determined to be suboptimal. We maintain a variable, bsf (best-so-far), which represents the minimum final distance among the candidates processed with Q so far. bsf serves as an upper-bound threshold. During the evaluation of a new candidate C, we monitor the accumulated partial distance, $dist_{\text{prev}}$. If $dist_{\text{prev}}$ exceeds bsf, the corresponding final distance is guaranteed to exceed bsf. In such cases, the current search branch is immediately terminated. The implementation of this pruning mechanism is detailed in lines 20–21 in Algorithm 4, which are colored in orange. In the case of top-k search, we must maintain the top-k final distances and use the k-th distance as the threshold.

Lower bound:

When we use DTW as a routine in PSD, the computation of the DTW of the interpolated segment can be sped up by computing LB_{Shen} for the lower bound. From Figure 4.7, we observe that a segment of Q, with length L^Q, is compared to a set of growing segments of C, with a fixed end at j. The length of these segments is from $L^C_{\min}$ to $L^C_{\max}$, as indicated in line 14 in Algorithm 4. They share the same suffix with length $L^C_{\min}$. It encourages us to view both Q and C reversely. The reversed segments are denoted as Q' and C', as in lines 6 and 16 in Algorithm 4. In this reversed view, they share the same prefix with length $L^C_{\min}$. Hence, we construct an indexed collection $\tilde{\mathbb{Q}}$ for Q' with the maximum length of the segments being compared in C, which is $L^C_{\max}$, as in lines 10–12.

In lines 24–27, we first compute the lower bound between Q' and the shortest segment of C. We add the minimum possible contribution of each c' to the distance contributed by the first alignment, which is $lb = (c'_1 - q'_1)$. For the lower bound of the subsequent segments, we compute them incrementally in line 29. Because the last point of segment C' must map to the last point of Q', we further tighten the lower bound by using the last alignment in line 30 instead.

Furthermore, we can reduce the computational overhead of constructing the sorted reaches $\tilde{\mathbb{q}}_k$ (lines 9–12). As illustrated in Figure 4.7, the segment of Q involved in the comparison grows incrementally. Since the reversed versions of these segments share a common prefix, the sorted reaches $\tilde{\mathbb{q}}_k$ computed for a segment Q' of length L^Q can be reused to compute those for the subsequent segments. This optimization is detailed in Algorithm 5, with the new components highlighted in blue. It is important to note that because r' is a function of L^Q, and the construction of $\tilde{\mathbb{q}}_k$ depends on r, reuse is limited to cases where r' remains constant. We construct reaches $\tilde{\mathbb{q}}_k$ from the sketch in lines 20–22 and keep track of the r' used for construction in line 23. If r' remains constant, we can use previously computed reaches $\tilde{\mathbb{q}}_k$ to compute the new set of reaches $\tilde{\mathbb{q}}_k$. Since the ending index of reaches depends on $\min(\lfloor kl \rfloor + r', L^Q)$, we need to check whether we have a new ending index when L^Q increases. If the ending index has been changed, we need to add the new data points $q'_{e_{\text{new}}}$ to the sorted sequence $\tilde{\mathbb{q}}_k$, as in line 15.

$L^C_{\max}$ depends of L^Q. If $L^C_{\max}$ increase because L^Q increase, we construct the new reach $\tilde{\mathbb{q}}_k$ and sort it in lines 16–18.

4.4.3 Guided Distance

For faster computation, one would want to use a distance measure with linear complexity, such as ED, as the base measure for PSD. While PSED is effective for identifying phase-scaling changes, certain applications require capturing complex properties within those segments that ED cannot handle. There are two ways to address it. One approach is to use an alternative distance measure that captures these complex properties as the base distance for computing the PSD, such as DTW. But PSDTW is slower than PSED. The other approach is to use the segmentation result returned by PSED. To address this, we propose a two-stage framework in which PSED-derived segmentation guides the application of advanced distance metrics M. Let $\mathcal{P}^* = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{P+1}, j_{P+1})\}$ be the set of optimal cut points on Q and C obtained by minimizing the PSED. We utilize $\mathcal{P}^*$ to partition both series into P aligned pairs of segments. The final distance, denoted as M^{PSED}, is calculated by summing the distances of these pairs using a target metric M:

$$M^{\text{PSED}}(Q, C) = \sum_{p=1}^{P} M(Q_p^L, C_p^L) \tag{4.16}$$

, where $Q_p = Q(i_p + 1 : i_{p+1})$ and $C_p = C(j_p + 1 : j_{p+1})$.

4.5 Experiments

We evaluate the performance of our proposed distance measure framework, PSD, via its two instantiations: PSED and PSDTW. Specifically, we focus on a query retrieval task in which the query exhibits piecewise-scaled distortion, that is, distinct phases of the query exhibit different expression rates relative to the target in the candidate dataset. Our objective is to verify whether PSD achieves invariance to these multiple scaling distortions, thereby allowing it to correctly retrieve the most similar candidate. We detail the experimental setup in Section 4.5.1 and present the results in Section 4.5.2. The code and data are available at `https://github.com/colemanyu/k-scaling-factor-dtw`.

4.5.1 Experimental Setup

We utilize the GunPoint dataset, originally released in 2003 [50], as the running example. Widely regarded as the "iris" dataset of the time series community [15], it has appeared in over one thousand publications. Beyond its ubiquity, the dataset addresses a critical distinction: misinterpreting the act of aiming a gun as merely pointing a finger could have life-threatening consequences.

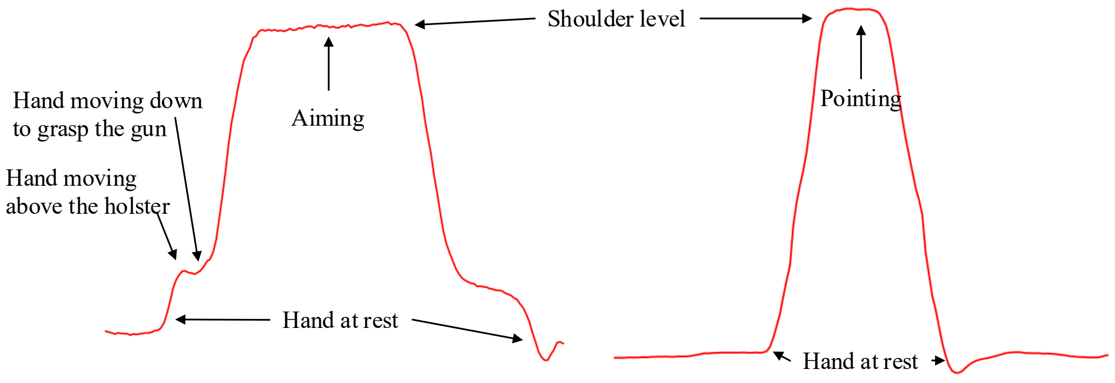


Figure 4.8: Visualization of the GunPoint dataset. Left (Right): A time series of the "Gun" ("Point") scenario. Critical periods, such as "Aiming", are annotated.

The dataset contains two classes: "Gun" and Point". Actors aim at an eye-level target using either a replica gun or their index finger, as illustrated in Figure 4.8. The resulting time series represent the X-axis centroid of the actor's right hand. Each action lasts for 5 seconds, with the pointing/aiming phase occurring for approximately one second. Recorded at 30 fps, each sample consists of 150 data points. The dataset comprises 50 training and 150 test series, all of length 150.

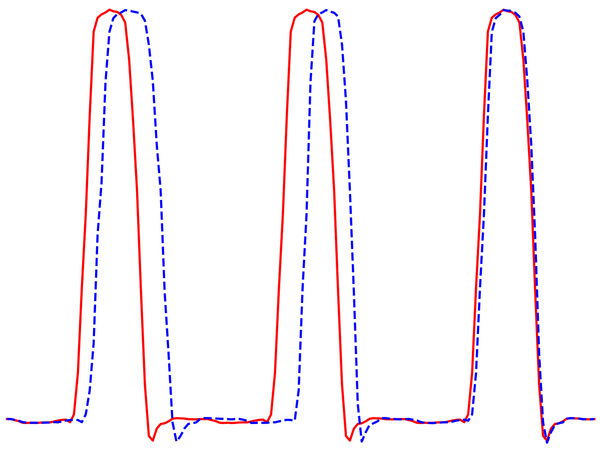


Figure 4.9: The red solid time series shows an instant in the target set. The blue dashed time series shows an instant in the query set.

A key limitation of the original dataset is the assumption that every action lasts exactly 5 seconds. In reality, actors perform actions at varying speeds. If an actor is asked to perform the action three times continuously in a row, we are likely to observe a time series containing three phases, each with a unique rate. The first phase should take longer than subsequent phases because it is the first time the action is performed. This phenomenon is depicted in Figure 4.9, where the red curve represents an ideal case that consists of three identical phases (i.e., identical rate), while the blue curve represents a realistic scenario with varying rates. The first action is slower than the second action. Consequently, retrieving such patterns requires assigning different scaling factors to each phase to accommodate the phase-specific rates.

We now describe the procedure for generating the target and query sets for the retrieval task. To construct the target set, we concatenate P repetitions of each time series instance from the source dataset. To ensure a fair comparison, we constrain the resulting time series to match the original length n. This is achieved by rescaling each phase to a length of n/P prior to concatenation. Note that we must handle remainders to ensure that the final time series length is exactly n. An example of such a target (where $P = 3$) is depicted by the red solid line in Figure 4.9.

To construct the query set, we first determine the specific lengths for the P segments. Starting with an expected mean length of n/P, we define the minimum segment length as $L_{\min} = (n/P)\sqrt{l}$ and the maximum as $L_{\max} = (n/P)\sqrt{l}$. This formulation ensures that the ratio of the lengths of any two segments is bounded by l. We then generate P random integers within $[L_{\min}, L_{\max}]$ subject to the constraint that their sum is exactly n. Finally, we construct the query by rescaling P copies of the source time series to these generated random lengths and concatenating them. The resulting time series maintains the original length n. An example of such a query (where $P = 3$) is depicted by the blue dashed line in Figure 4.9.

Both the query and target sets are derived from the same source dataset. Consequently, the ground truth target for a given query is defined as the instance in the target set that is generated from the same underlying source time series.

Table 4.1 details the additional datasets used in this study. The column labeled "Train/Test?" specifies which split was employed as the source dataset.

In our experiments, we set the warping window parameter $r = 0.1$, as suggested in the literature. The algorithms were implemented in Python. We utilized the `aeon` [43] library to obtain the baseline distance measures. All experiments were conducted on a workstation equipped with an Intel Xeon Gold 6326 CPU and 256GB of RAM.

Table 4.1: Details of the ten datasets for the experiments

Name	Type	Train/Test?	Size	Class	Length
SonyAIBORobotSurface1	Sensor	Test	601	2	70
ECG200	ECG	Train	100	2	96
MedicalImages	Image	Train	381	10	99
CBF	Simulated	Train	30	3	128
SwedishLeaf	HAR	Train	500	15	128
Plane	Sensor	Train	105	7	144
PowerCons	Device	Train	180	2	144
GunPoint	HAR	Train	50	2	150
Adiac	Image	Train	390	37	176
Epilepsy	HAR	Train	137	4	207

4.5.2 Experimental Results

We employ Top-k Accuracy (often referred to as Precision@k [61]) as the primary evaluation metric. For a single query Q, this metric indicates whether the correct match is successfully retrieved within the top k candidates:

$$P@k(Q) = \begin{cases} 1 & \text{if the ground truth is in the top-}k\text{ results} \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

To determine the top-k results, we compute the distance between Q and every time series in the target set, generating a distance profile of length equal to the dataset size. The top-k results correspond to the k instances with the smallest distances in this profile. Finally, we evaluate the overall performance by computing the mean Top-k Accuracy across the entire query set $\mathcal{D}$:

$$\overline{P@k} = \frac{\sum_{Q\in\mathcal{D}} P@k(Q)}{|\mathcal{D}|} \tag{4.18}$$

We choose $k \in 1, 3$. $k = 1$ refers to the exact retrieval. $k = 3$ give some tolerance for the retrieval.

Table 4.2: The accuracy comparison for eight distance measures of the GunPoint dataset

P	l	ED		DTW [53]		ADTW [26]		DDTW [32]		shapeDTW [66]		WDDTW [29]		WDTW
		P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1
2	1.25	0.30	0.56	0.82	1.00	0.54	0.84	0.84	0.92	0.96	1.00	0.88	0.96	0.82
	1.50	0.14	0.36	0.88	1.00	0.38	0.64	0.78	0.94	**1.00**	1.00	0.80	0.92	0.88
	1.75	0.16	0.34	0.82	1.00	0.38	0.66	0.64	0.80	0.90	1.00	0.66	0.80	0.82
	2.00	0.12	0.24	0.84	0.98	0.34	0.66	0.72	0.88	0.96	1.00	0.68	0.86	0.82
3	1.25	0.30	0.50	0.84	0.92	0.70	0.88	0.80	0.92	**0.96**	0.98	0.84	0.94	0.86
	1.50	0.10	0.28	0.84	0.96	0.60	0.78	0.66	0.86	0.90	1.00	0.72	0.86	0.84
	1.75	0.10	0.28	0.88	1.00	0.42	0.78	0.72	0.88	0.76	0.94	0.74	0.88	0.88
	2.00	0.02	0.12	0.86	0.94	0.38	0.52	0.60	0.78	0.70	0.94	0.60	0.80	0.82
4	1.25	0.34	0.50	0.70	0.86	0.68	0.88	0.60	0.78	0.70	0.90	0.64	0.80	0.70
	1.50	0.22	0.38	0.64	0.80	0.60	0.92	0.52	0.64	0.68	0.90	0.52	0.66	0.64
	1.75	0.16	0.30	0.72	0.92	0.56	0.80	0.60	0.78	0.56	0.80	0.56	0.80	0.70
	2.00	0.06	0.12	0.58	0.78	0.50	0.72	0.46	0.64	0.50	0.82	0.50	0.64	0.56

Results of GunPoint dataset: We utilize the GunPoint dataset to evaluate how the parameters P and l affect the ability of PSD to achieve invariance under piecewise scaling. The results are presented in Table 4.2, where the best performance of $\overline{P@1}$ is highlighted in bold, and the second-best is underlined. We benchmark our method against several state-of-the-art distance measures, including ADTW [26], DDTW [32], shapeDTW [66], WDDTW [29], and WDTW [29],. PSED achieves the highest accuracy, followed closely by PSDTW. We attribute PSED's superior performance over PSDTW to the specific nature of the distortions in the query set, that is, the "pure" piecewise scaling distortions. Since the query set exhibits only piecewise scaling distortions, the corresponding segments of the query and target time series differ solely in length (scale). Consequently, the additional flexibility provided by DTW in PSDTW is unnecessary and may inadvertently increase the similarity of incorrect matches, thereby reducing discriminative power relative to the stricter PSED measure. However, PSDTW remains theoretically essential for handling local distortions within the phase. PSDTW applies DTW within the scaled segment, enabling robust alignment across local nonlinearities.

Table 4.3: The accuracy comparison for six PSED-guided distance measures of the GunPoint dataset

P	l	DTW^{PSED}		$\text{ADTW}^{\text{PSED}}$		$\text{DDTW}^{\text{PSED}}$		$\text{shapeDTW}^{\text{PSED}}$		$\text{WDDTW}^{\text{PSED}}$		$\text{WDTW}^{\text{PSED}}$	
		P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3
2	1.25	0.86	1.00	0.98	1.00	**0.72**	0.92	0.98	1.00	**0.74**	0.96	0.86	1.00
	1.50	0.88	1.00	1.00	1.00	**0.60**	0.90	**0.98**	1.00	**0.64**	0.90	0.92	1.00
	1.75	0.94	1.00	1.00	1.00	0.84	0.94	1.00	1.00	0.86	0.96	0.94	1.00
	2.00	0.98	1.00	1.00	1.00	**0.60**	0.78	0.98	1.00	**0.66**	0.80	0.98	1.00
3	1.25	**0.82**	0.96	0.94	1.00	**0.74**	0.90	**0.92**	0.98	**0.76**	0.90	**0.82**	0.96
	1.50	0.86	1.00	0.96	1.00	0.72	0.90	0.94	1.00	0.80	0.90	0.88	1.00
	1.75	0.88	1.00	1.00	1.00	**0.68**	0.90	1.00	1.00	**0.70**	0.94	0.90	1.00
	2.00	0.86	0.96	0.98	1.00	**0.54**	0.86	0.96	1.00	0.62	0.86	0.88	0.96
4	1.25	0.74	0.88	0.86	0.94	0.78	0.80	0.80	0.92	0.78	0.82	0.74	0.88
	1.50	0.72	0.84	0.86	0.98	0.68	0.84	0.84	0.96	0.66	0.84	0.72	0.86
	1.75	0.74	0.96	0.92	1.00	**0.54**	0.90	0.90	1.00	0.56	0.88	0.74	0.96
	2.00	0.66	0.90	0.88	0.98	0.58	0.82	0.88	0.98	0.62	0.80	0.66	0.90

We further investigate whether the segmentation results returned by PSED can serve as a guide to enhance other distance measures. As shown in Table 4.3, this approach generally improves accuracy. The exceptions are highlighted in bold, indicating cases where the segmentation led to worse performance. Notably, in most cases, only DDTW and WDDTW failed to benefit from PSED-guided segmentation. We argue that the performance degradation of DDTW and WDDTW stems from their derivative-based nature. They rely on matching slope or shape features.

Consequently, they are highly sensitive to segmentation boundaries. A non-perfect cut that falls within a shape feature segmentize it, and these features are then destroyed. When the algorithm subsequently attempts to map these features, it fails to find correct correspondences, resulting in high distances.

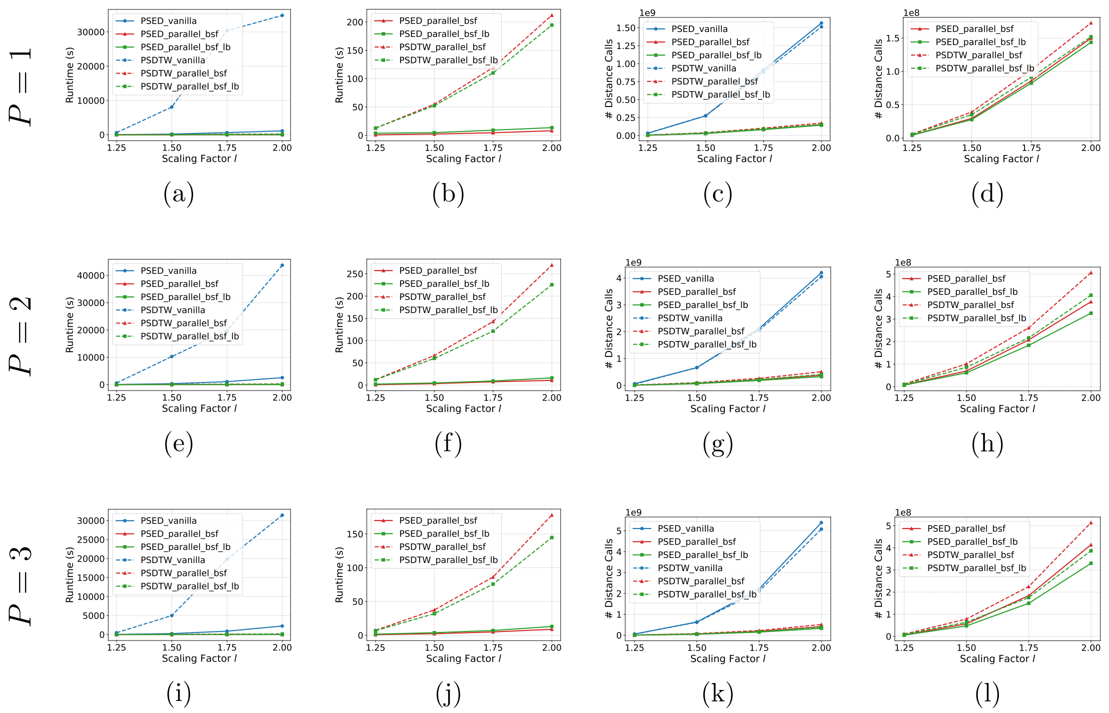


Figure 4.10: Runtime and number of distance calls comparison of GunPoint dataset. (a)-(d) $P = 1$, (e)-(h) $P = 2$, (i)-(l) $P = 3$.

Figure 4.10 illustrates the runtime performance across varying parameters P and l. We evaluate two variants of PSD, PSED and PSDTW, each implemented with three levels of

1. `vanilla` (i.e., Basic implementation)

2. `parallel_bsf` (i.e., With parallelization with Best-So-Far early abandoning)

3. `parallel_bsf_lb` (i.e., Incorporating lower bound pruning)

This yields a total of six methods. The figure is organized into four columns plotted against the scaling factor l. The rows refer to the number of pieces P. The first and third columns display the running time and the number of distance calculations, respectively, for all six methods. To better visualize the performance differences among the efficient implementations, the second and fourth columns omit the `vanilla` baselines and focus exclusively on the four optimized variants.

The results indicate that `vanilla_PSDTW` is orders of magnitude slower than the other approaches, whereas the optimized methods operate within a similar performance range. As anticipated, the computation time for all methods increases with the scaling factor l. The fourth column confirms that the lower bounding strategy effectively reduces the number of distance calculations (pruning power). However, the second column reveals a critical trade-off in actual runtime. While the lower bound successfully accelerates PSDTW, it actually slows down PSED. This suggests that for the computationally lighter ED, the overhead of calculating the lower bound outweighs the time saved by pruning. Overall, the results demonstrate that PSED is significantly faster than PSDTW.

Results of the ten datasets:

For the remaining nine datasets, we fix the parameters at $P = 3$ and $l = 1.5$. We have the following findings from the previous experiment:

1. From Table 4.3, PSED outperformed PSDTW in handling piecewise scaling distortions.

2. From Figure 4.10, the lower bound offered no efficiency gain for PSED.

Hence, we select `PSED_parallel_bsf` as the representative method for this evaluation. The accuracy results are presented in Table 4.4, while the results for

Table 4.4: The accuracy comparison for eight distance measures of the ten datasets

Dataset	ED		DTW		ADTW		DDTW		shapeDTW		WDDTW	
	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3
SonyAIBORobotSurface1	0.04	0.08	0.51	0.63	$\underline{0.60}$	0.74	0.45	0.57	0.20	0.29	0.45	0.58
ECG200	0.12	0.20	0.70	0.81	$\underline{0.78}$	0.83	0.60	0.74	0.59	0.76	0.59	0.73
MedicalImages	0.08	0.19	0.63	0.78	0.69	0.85	0.52	0.68	0.57	0.75	0.51	0.68
CBF	0.53	0.67	0.83	1.00	$\underline{0.90}$	1.00	0.73	0.87	0.73	0.90	0.77	0.90
SwedishLeaf	0.07	0.12	0.82	0.92	$\underline{0.84}$	0.93	0.70	0.83	0.78	0.91	0.69	0.83
Plane	0.09	0.14	0.50	0.74	$\underline{0.59}$	0.78	0.38	0.64	0.53	0.79	0.37	0.61
PowerCons	0.26	0.43	0.77	0.98	**0.80**	1.00	0.77	0.98	0.77	0.99	0.77	0.99
GunPoint	0.10	0.28	$\underline{0.84}$	0.96	0.60	0.78	0.66	0.86	0.90	1.00	0.72	0.86
Adiac	0.01	0.02	$\underline{0.37}$	0.50	0.12	0.21	0.32	0.42	0.27	0.41	0.31	0.41
Epilepsy	0.18	0.26	0.74	0.88	$\underline{0.80}$	0.91	0.65	0.82	0.38	0.47	0.58	0.79

Table 4.5: The accuracy comparison for six PSED-guided distance measures of the ten datasets

Dataset	DTW$^{\text{PSED}}$		ADTW$^{\text{PSED}}$		DDTW$^{\text{PSED}}$		shapeDTW$^{\text{PSED}}$		WDDTW$^{\text{PSED}}$		W
	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P@1	P@3	P
SonyAIBORobotSurface1	0.53	0.64	0.64	0.76	**0.37**	0.53	0.65	0.76	**0.38**	0.53	0
ECG200	0.73	0.84	0.80	0.88	**0.54**	0.68	0.81	0.88	**0.53**	0.69	0
MedicalImages	0.66	0.78	0.76	0.88	0.51	0.69	0.74	0.88	0.51	0.70	0
CBF	0.90	1.00	0.93	1.00	0.73	0.93	0.93	0.97	**0.70**	0.93	0
SwedishLeaf	0.82	0.92	0.97	0.99	0.72	0.87	0.97	0.99	0.73	0.88	**0**
Plane	0.60	0.80	0.75	0.94	0.50	0.67	0.74	0.90	0.50	0.69	0
GunPoint	0.86	1.00	0.96	1.00	0.72	0.90	0.94	1.00	0.80	0.90	0
PowerCons	0.78	0.99	0.79	1.00	0.79	1.00	0.79	1.00	0.79	1.00	0
Adiac	**0.33**	0.45	0.57	0.71	**0.21**	0.34	0.53	0.70	**0.20**	0.33	**0**
Epilepsy	0.77	0.88	**0.78**	0.88	0.66	0.79	0.74	0.89	0.66	0.82	0

the PSED-guided distance measures are detailed in Table 4.5. Finally, the runtime efficiency for all datasets is summarized in Table 4.6. It shows a significant speedup, ranging from 10.10X to 191.46X.

Table 4.6: The number of distance calls and runtime on PSED_vanilla and PSED_parallel_bsf of the ten datasets

Name	Size	Length	PSED_vanilla	PSED_parallel_bsf	% distance call
			Time (s)	Time (s)	
SonyAIBORobotSurface1	601	70	697	69	90.40%
ECG200	100	96	91	3	91.08%
MedicalImages	381	99	2082	40	96.03%
CBF	30	128	43	2	75.31%
SwedishLeaf	500	128	16601	107	96.24%
Plane	105	144	621	6	96.05%
PowerCons	180	144	3629	49	82.32%
GunPoint	50	150	159	3	89.42%
Adiac	390	175	23550	123	96.71%
Epilepsy	137	206	13131	336	39.88%

4.6 Conclusion and Future Work

In this paper, we proposed a novel distance measure framework, Piecewise Scaling Distance (PSD), which relaxes the strict assumption of a single uniform scaling factor across the entire time series. We presented an exact dynamic programming (DP) algorithm to solve this problem. To enhance efficiency and prevent pathological segment alignments, we introduced a constraint version, which limits the search space of segment lengths based on given scaling factors. To enhance computational efficiency, we integrated two optimization techniques for the general PSD framework. In addition, we propose incorporating a lower-bounding strategy to accelerate PSDTW. Our experimental results demonstrate the necessity and effectiveness of PSD when identifying matches between a query Q and a candidate C under piecewise scaling distortions.

We outline several directions for future research. First, we aim to develop a lower bound specifically optimized for PSED that can improve actual runtime.

Second, while the current PSDTW algorithm requires the number of segments P to be specified as a hyperparameter, it is preferable for the algorithm to determine this value adaptively. A simple heuristic is to test a range of P values and select the configuration that yields the minimum distance Finally, we plan to investigate efficiency improvements for PSDTW. Currently, PSDTW computes dynamic time warping on two growing subsequences after scaling. While techniques for Incremental DTW [35] allow for the reuse of the accumulating cost matrix D to avoid redundant calculations, applying this to PSDTW is non-trivial. The interpolation performed before DTW fundamentally alters the subsequence structure, preventing the straightforward extension of D (e.g., by appending rows or columns) to reuse the computed D that is possible in standard DTW.

Algorithm 4 Constrainted PSDTW (cPSDTW) with early abandoning and lower bounding

```
Input: Query series $Q$, Candidate series $C$, DTW constraint parameter $r$ (in fraction), Number of pieces $P$, best_so_far bsf
Output: The final distance $D[m,n,P]$ if $D[m,n,P] \leq$ bsf, otherwise $\infty$
 1: Execute Algorithm 3 for initialization
 2: for $p \leftarrow 1$ to $P$ do
 3:     for $i \leftarrow (p \cdot L^Q_{\text{gmin}})$ to $\min(p \cdot L^Q_{\text{gmax}}, m)$ do    ▷ The iterations can be parallelized.
 4:         for $L^Q \leftarrow L^Q_{\text{gmin}}$ to $L^Q_{\text{gmax}}$ do
 5:             $i' \leftarrow i - L^Q$    ▷ $i'$: End point of previous segment on $Q$.
 6:             $Q' \leftarrow \text{rev}(Q(i'+1 : i))$    ▷ rev($T$): Reverse the input series $T$.
 7:             $L^C_{\text{min}} \leftarrow \max(L^C_{\text{gmin}}, \lceil L^Q/l \rceil)$
 8:             $L^C_{\text{max}} \leftarrow \min(\lfloor L^Q/l \rfloor, L^C_{\text{gmax}})$
 9:             $r' \leftarrow r \times \max(L^Q, L^C_{\text{max}})$    ▷ Compute the integer value of $r$.
10:             for $k \leftarrow 1$ to $L^C_{\text{max}}$ do
11:                 $\mathbb{q}_k \leftarrow (q'_{\max(1,\lceil k/l \rceil - r')}, \ldots, q'_{\min(\lfloor kl \rfloor + r', L^Q)})$    ▷ Construct indexed collection $\mathbb{Q}$.
12:                 $\tilde{\mathbb{q}}_k \leftarrow \text{sort}(\mathbb{q}_k)$
13:                 for $j \leftarrow (p \cdot L^C_{\text{gmin}})$ to $\min(p \cdot L^C_{\text{gmax}}, n)$ do
14:                     for $L^C \leftarrow L^C_{\text{min}}$ to $L^C_{\text{max}}$ do
15:                         $j' \leftarrow j - L^Q$
16:                         $C' \leftarrow \text{rev}(C(j'+1 : j))$
17:                         $dist_{\text{prev}} \leftarrow D[i', j', p-1]$
18:                         if $dist_{\text{prev}} = \infty$ then
19:                             continue
20:                         if $dist_{\text{prev}} >$ bsf then
21:                             continue
22:                         if $dist_{\text{prev}} > D[i,j,p]$ then    ▷ $D[i,j,p]$ stores the best so far.
23:                             continue
24:                         if $L^C = L^C_{\text{min}}$ then    ▷ Compute the lower bound from sketch.
25:                             $lb = (c'_1 - q'_1)^2$    ▷ Partial distance contributed by the first alignment.
26:                             for $k \leftarrow 2$ to $L^C$ do
27:                                 $lb = lb + \delta(c'_k, \tilde{\mathbb{q}}_k)$
28:                         else
29:                             $lb = lb + \delta(c'_{L^C}, \tilde{\mathbb{q}}_{L^C})$    ▷ Compute the lower bound incrementally.
30:                         $lb_{\text{check}} = lb - \delta(c'_{L^C}, \tilde{\mathbb{q}}_{L^C}) + (c'_{-1} - q'_{-1})^2$    ▷ Tighten the lower bound by using the last alignment.
31:                         if $dist_{\text{prev}} + lb_{\text{check}} > D[i,j,p]$ then
32:                             continue
33:                         $dist_{\text{seg}} \leftarrow \text{DTW}_r(Q'^L, C'^L)$
34:                         if $dist_{\text{prev}} + dist_{\text{seg}} < D[i,j,p]$ then
35:                             $D[i,j,p] \leftarrow dist_{\text{prev}} + dist_{\text{seg}}$    ▷ Also save pointer $(i', j')$ for the cut.
36: return $D[m,n,P]$
```

Algorithm 5 Replace lines 3 to 9 in Algorithm 5to reuse the computed sorted reaches $\tilde{\mathbb{q}}_k$.

```
1: $r'_{\text{cache}} \leftarrow -1$
2: **for** $i \leftarrow (p \cdot L^Q_{\text{gmin}})$ **to** $\min(p \cdot L^Q_{\text{gmax}}, m)$ **do**
3:     **for** $L^Q \leftarrow L^Q_{\text{gmin}}$ **to** $L^Q_{\text{gmax}}$ **do**
4:         $i' \leftarrow i - L^Q$
5:         $Q' \leftarrow \text{rev}(Q(i'+1:i))$
6:         $L^C_{\min} \leftarrow \max(L^C_{\text{gmin}}, \lceil L^Q/l \rceil)$
7:         $L^C_{\max} \leftarrow \min(\lfloor L^Q/l \rfloor, L^C_{\text{gmax}})$
8:         $r' \leftarrow r \times \max(L^Q, L^C_{\max})$
9:         **if** $r'_{\text{cache}} = r'$ **then**
10:            **for** $k \leftarrow 1$ **to** $L^C_{\max}$ **do**
11:                **if** $k \leq |\mathbb{Q}|$ **then**
12:                    $e_{\text{prev}} \leftarrow \min(\lfloor kl \rfloor + r', L^Q - 1)$
13:                    $e_{\text{new}} \leftarrow \min(\lfloor kl \rfloor + r', L^Q)$
14:                    **if** $e_{\text{new}} > e_{\text{prev}}$ **then**
15:                        $\tilde{\mathbb{q}}_k \leftarrow \tilde{\mathbb{q}}_k \cup q'_{e_{\text{new}}}$
16:                **else**
17:                    $\mathbb{q}_k \leftarrow (q'_{\max(1,\lceil k/l \rceil - r')}, \ldots,$
                                   $q'_{\min(\lfloor kl \rfloor + r', L^Q)})$          ▷ $L^Q$ equals to $|Q'|$.
18:                    $\tilde{\mathbb{q}}_k \leftarrow \text{sort}(\mathbb{q}_k)$
19:        **else**
20:            **for** $k \leftarrow 1$ **to** $L^C_{\max}$ **do**
21:                $\mathbb{q}_k \leftarrow (q'_{\max(1,\lceil k/l \rceil - r')}, \ldots,$
                                   $q'_{\min(\lfloor kl \rfloor + r', L^Q)})$
22:                $\tilde{\mathbb{q}}_k \leftarrow \text{sort}(\mathbb{q}_k)$
23:            $r_{\text{cache}} \leftarrow r'$
```

Chapter 5

Conclusion

Bibliography

[1] Charu C. Aggarwal. *Data Mining: The Textbook*. Cham, 2015.

[2] Charu C. Aggarwal and Chandan K. Reddy, editors. *Data Clustering: Algorithms and Applications*. Boca Raton, 1st edition edition, August 2013.

[3] Firoz Ahmed, Rakesh Kaundal, and Gajendra PS Raghava. Phdcleav: A svm based method for predicting human dicer cleavage sites using sequence and secondary structure of mirna precursors. *BMC Bioinformatics*, 14(14):S9, October 2013.

[4] Mahmood Akhtar, Julien Epps, and Eliathamby Ambikairajah. On dna numerical representations for period-3 based exon prediction. In *2007 IEEE International Workshop on Genomic Signal Processing and Statistics*, pages 1–4, June 2007.

[5] Bruce Alberts. *Molecular Biology of the Cell*. New York, seventh edition edition, 2022.

[6] D. Anastassiou. Genomic signal processing. *IEEE Signal Processing Magazine*, 18(4):8–20, July 2001.

[7] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017.

[8] Yu Bao, Morihiro Hayashida, and Tatsuya Akutsu. Lbsizecleav: Improved support vector machine (svm)-based prediction of dicer cleavage sites using loop/bulge length. *BMC Bioinformatics*, 17(1):487, November 2016.

[9] Gustavo E. A. P. A. Batista, Eamonn J. Keogh, Oben Moses Tataw, and Vinícius M. A. de Souza. Cid: An efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, 28(3):634–669, May 2014.

[10] John A Berger, Sanjit K Mitra, Marco Carli, and Alessandro Neri. Visualization and analysis of dna sequences using dna walks. *Journal of the Franklin Institute*, 341(1):37–53, January 2004.

[11] Niranjan Chakravarthy, A. Spanias, L. D. Iasemidis, and K. Tsakalis. Autoregressive modeling and feature analysis of dna sequences. *EURASIP Journal on Advances in Signal Processing*, 2004(1):1–16, December 2004.

[12] William W. Cohen. *A Computer Scientist's Guide to Cell Biology: A Travelogue from a Stranger in a Strange Land.* New York, NY, 2007.

[13] P. D. Cristea. Conversion of nucleotides sequences into genomic signals. *Journal of Cellular and Molecular Medicine*, 6(2):279–303, 2002.

[14] Roger B. Dannenberg, William P. Birmingham, George P. Tzanetakis, Colin P. Meek, Ning P. Hu, and Bryan P. Pardo. The musart testbed for query-by-humming evaluation. *Comput. Music J.*, 28(2):34–48, June 2004.

[15] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, November 2019.

[16] Angus Dempster, François Petitjean, and Geoffrey I. Webb. Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, September 2020.

[17] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 248–257, New York, NY, USA, August 2021.

[18] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5):1779–1805, September 2023.

[19] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Computing Surveys*, 45(1):1–34, November 2012.

[20] Yong Feng, Xiaoxiao Zhang, Paul Graves, and Yan Zeng. A comprehensive analysis of precursor microrna cleavage by human dicer. *RNA*, 18(11):2083–2092, November 2012.

[21] Ada Wai-Chee Fu, Eamonn Keogh, Leo Yung Hang Lau, Chotirat Ann Ratanamahatana, and Raymond Chi-Wing Wong. Scaling and time warping in time series querying. *The VLDB Journal*, 17(4):899–921, July 2008.

[22] Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, February 2011.

[23] J. Gorodkin. Comparing two k-category assignments by a k-category correlation coefficient. *Computational Biology and Chemistry*, 28(5):367–374, December 2004.

[24] Sam Griffiths-Jones, Harpreet Kaur Saini, and Stijn van Dongen. mirbase: Tools for microrna genomics. *Nucleic Acids Research*, 36(suppl_1):D154–D158, January 2008.

[25] Shuo Gu, Lan Jin, Yue Zhang, Yong Huang, Feijie Zhang, Paul N. Valdmanis, and Mark A. Kay. The loop position of shrnas and pre-mirnas is critical for the accuracy of dicer processing in vivo. *Cell*, 151(4):900–911, November 2012.

[26] Matthieu Herrmann and Geoffrey I. Webb. Amercing: An intuitive and effective constraint for dynamic time warping. *Pattern Recognition*, 137:109333, May 2023.

[27] Todd Holden, R. Subramaniam, R. Sullivan, E. Cheung, C. Schneider, G. Tremberger Jr, A. Flamholz, D. H. Lieberman, and T. D. Cheung. Atcg nucleotide fluctuation of deinococcus radiodurans radiation genes. In *Instruments, Methods, and Missions for Astrobiology X*, volume 6694, pages 402–411, October 2007.

[28] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, February 1975.

[29] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9):2231–2240, September 2011.

[30] Eamonn Keogh, Themistoklis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. Indexing large human-motion databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 780–791, Toronto, Canada, August 2004.

[31] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, March 2005.

[32] Eamonn J. Keogh and Michael J. Pazzani. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM International Conference on Data Mining (SDM)*, Proceedings, pages 1–11, April 2001.

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

[34] Yoontae Lee, Kipyoung Jeon, Jun-Tae Lee, Sunyoung Kim, and V. Narry Kim. Microrna maturation: Stepwise processing and subcellular localization. *The EMBO Journal*, 21(17):4663–4670, September 2002.

[35] Maximilian Leodolter, Claudia Plant, and Norbert Brändle. Incdtw: An r package for incremental calculation of dynamic time warping. *Journal of Statistical Software*, 99:1–23, September 2021.

[36] Ker-Chau Li, Ming Yan, and Shinsheng Yuan. A simple statistical model for depicting the cdc15-synchronized yeast cell-cycle regulated gene expression data. *Statistica Sinica*, 12(1):141–158, 2002.

[37] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: A two-level statistical model for character motion synthesis. *ACM Trans. Graph.*, 21(3):465–472, July 2002.

[38] Pengyu Liu, Jiangning Song, Chun-Yu Lin, and Tatsuya Akutsu. Recgbm: A gradient boosting-based method for predicting human dicer cleavage sites. *BMC Bioinformatics*, 22(1):63, February 2021.

[39] Ronny Lorenz, Stephan H. Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. Viennarna package 2.0. *Algorithms for Molecular Biology*, 6(1):26, November 2011.

[40] Ian J. MacRae, Kaihong Zhou, and Jennifer A. Doudna. Structural determinants of rna recognition and cleavage by dicer. *Nature Structural & Molecular Biology*, 14(10):934–940, October 2007.

[41] B. W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, October 1975.

[42] Gerardo Mendizabal-Ruiz, Israel Román-Godínez, Sulema Torres-Ramos, Ricardo A. Salido-Ruiz, and J. Alejandro Morales. On dna numerical representations for genomic similarity computation. *PLOS ONE*, 12(3):e0173288, March 2017.

[43] Matthew Middlehurst, Ali Ismail-Fawaz, Antoine Guillaume, Christopher Holder, David Guijo-Rubio, Guzal Bulatova, Leonidas Tsaprounis, Lukasz Mentel, Martin Walter, Patrick Schäfer, and Anthony Bagnall. Aeon: A python toolkit for learning from time series. *Journal of Machine Learning Research*, 25(289):1–10, 2024.

[44] Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: A review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, 38(4):1958–2031, July 2024.

[45] Lixuan Mu and Tatsuya Akutsu. Dicleaveplus: A transformer-based model to detect human dicer cleavage sites within cleavage patterns. *Genes to Cells*, 31(1):e70074, 2026.

[46] Lixuan Mu, Jiangning Song, Tatsuya Akutsu, and Tomoya Mori. Dicleave: A deep learning model for predicting human dicer cleavage sites. *BMC Bioinformatics*, 25(1):13, January 2024.

[47] Achuthsankar S Nair and Sivarama Pillai Sreenadhan. A coding measure scheme employing electron-ion interaction pseudopotential (eiip). *Bioinformation*, 1(6):197–202, October 2006.

[48] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn 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*, KDD '12, pages 262–270, New York, NY, USA, August 2012.

[49] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Trans. Knowl. Discov. Data*, 7(3):10:1–10:31, September 2013.

[50] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *3 Rd International Workshop on Mining Temporal and Sequential Data (TDM-04)*, 2004.

[51] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 147–154, New York, NY, USA, August 1996.

[52] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, March 2021.

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

[54] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, October 2007.

[55] Yilin Shen, Yanping Chen, Eamonn Keogh, and Hongxia Jin. Searching time series with invariance to large amounts of uniform scaling. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 111–114, April 2017.

[56] Yilin Shen, Yanping Chen, Eamonn Keogh, and Hongxia Jin. Accelerating time series searching with large uniform scaling. In *Proceedings of the 2018 SIAM International Conference on Data Mining (SDM)*, Proceedings, pages 234–242, May 2018.

[57] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I. Webb. Elastic similarity and distance measures for multivariate time series. *Knowledge and Information Systems*, 65(6):2665–2698, June 2023.

[58] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I. Webb. Multirocket: Multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36(5):1623–1646, September 2022.

[59] Chang Wei Tan, Matthieu Herrmann, Germain Forestier, Geoffrey I. Webb, and François Petitjean. Efficient search of the best warping window for dy-

namic time warping. In *Proceedings of the 2018 SIAM International Conference on Data Mining (SDM)*, Proceedings, pages 225–233, May 2018.

[60] Chang Wei Tan, François Petitjean, and Geoffrey I. Webb. Elastic bands across the path: A new framework and method to lower bound dtw. In *Proceedings of the 2019 SIAM International Conference on Data Mining (SDM)*, Proceedings, pages 522–530, May 2019.

[61] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and recall for time series. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[62] Lisa A. Urry, Michael L. Cain, Steven Alexander Wasserman, Peter V. Minorsky, Rebecca B. Orr, and Neil A. Campbell. *Campbell Biology*. New York, NY, twelfth edition edition, 2020.

[63] Taosheng Xu, Ning Su, Lin Liu, Junpeng Zhang, Hongqiang Wang, Weijia Zhang, Jie Gui, Kui Yu, Jiuyong Li, and Thuc Duy Le. mirbaseconverter: An r/bioconductor package for converting and retrieving mirna name, accession, sequence and family information in different versions of mirbase. *BMC Bioinformatics*, 19(19):514, December 2018.

[64] Dragomir Yankov, Eamonn Keogh, Jose Medina, Bill Chiu, and Victor Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 844–853, New York, NY, USA, August 2007.

[65] Ren Zhang and Chun-Ting Zhang. Z curves, an intutive tool for visualizing and analyzing the dna sequences. *Journal of Biomolecular Structure and Dynamics*, 11(4):767–782, February 1994.

[66] Jiaping Zhao and Laurent Itti. shapedtw: Shape dynamic time warping. *Pattern Recognition*, 74:171–184, February 2018.

[67] Jing Zhao, Xiu Wen Yang, Jian Ping Li, and Yuan Yan Tang. Dna sequences classification based on wavelet packet analysis. In *Wavelet Analysis and Its Applications*, pages 424–429, 2001.

[68] Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 181–192, New York, NY, USA, June 2003.

[69] Marketa J. Zvelebil, Jeremy O. Baum, and Marketa Zvelebil. *Understanding Bioinformatics*. New York, 2008.