ONLINE EFFICIENT AND EFFECTIVE SEARCH IN LARGE AND NOISY SEQUENCE DATABASES

by

ALEXIOS KOTSIFAKOS

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2014

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Vassilis Athitsos for his invaluable advice and support during the course of my doctoral studies. I also wish to thank the professor Dr. Panagiotis Papapetrou who co-supervised my thesis, as well as my committee members Dr. Leonidas Fegaras, Dr. Chengkai Li, and Dr. Ramez Elmasri for their interest in my research, their helpful comments, and for taking time to serve in my dissertation committee.

I am grateful to all the teachers who taught me during all the years I spent in school, in Athens (Greece) and at the University of Texas at Arlington (UTA). In addition, I would like to thank my friends and colleagues for their confidence and trust on the successful completion of my doctoral studies.

Finally, I would like to express my deep gratitude to my family who have encouraged, inspired, and supported my undergraduate and graduate studies. This thesis is dedicated to my grandfathers and grandmothers, Evangelos, Ioannis-Prodromos, Eleni, and Athena, the spirit, blessings, wisdom, and light of whom were, are, and will always be leading my way.

April 4, 2014

ABSTRACT

ONLINE EFFICIENT AND EFFECTIVE SEARCH
IN LARGE AND NOISY SEQUENCE DATABASES

Alexios Kotsifakos, Ph.D.

The University of Texas at Arlington, 2014

Supervising Professor: Vassilis Athitsos

This thesis investigates the problem of similarity search in large and noisy sequence databases. A key application domain of interest in this work is the very challenging Query-By-Humming (QBH) problem, according to which, given a hummed part of a song, we would like to identify the closest matches in a large music repository. The problem of selecting the most appropriate, based on each specific query, distance measure out of a pool of measures for classification in time series data is also investigated. In addition, searching time series databases via examples, which may be either time series or models, is also part of this work. Moreover, motivated by the nature of music notes in the noisy QBH application, we explore the problem of interval-based sequence matching in a variety of application domains.

More specifically, this thesis makes contributions both by defining novel similarity measures that are used to identify the best database matches, and by proposing methods to improve efficiency. Referring to the similarity measures, the thesis contributes a method, named SMBGT (shorthand for Subsequence Matching with Bounded Gaps and Tolerances), for finding the closest subsequence matches from a

large database to a given query. This measure is applied to the noisy QBH application domain, where the music pieces are represented as 2-dimensional time series. Since efficiency should be a key characteristic when searching large time series databases, ISMBGT is also proposed, which performs indexing on top of the SMBGT method in a filter-and-refine manner. Applying ISMBGT on synthetic and real query sets for QBH shows the usefulness of our indexing scheme.

A second contribution of this thesis is the "Hum-a-song" QBH system, which allows the user to select among a variety of distance measures and music representations in order to retrieve the closest songs to a hummed query. Apart from that, SMBGT is also exploited to perform genre classification of music pieces, which, among others, can be beneficial in the area of assistive environments. For example, music systems with therapeutic and educational functionalities need to be adapted based on the music preferences of the end-users, such as children, patients, or disabled. Being able to identify the genres of songs that fit to each category of end-users is imperative in such settings as it can assist effectively in medical treatments and learning tasks.

Moreover, we address the problem of selecting the most appropriate distance measure out of a pool of measures, which depends on each specific query, for classification of time series. The proposed approach is, to the best of our knowledge, the first one to deal with this problem. The reason for the importance of such a problem is given through an example. Assuming that there are two available distance measures performing whole sequence matching, there may be cases where the first one may correctly classify a given query, while it may fail to classify another query, which is correctly classified by the second distance measure. Thus, it would be highly desirable to be able to choose the "best" measure for each query. The classification accuracy of the proposed method compared to three state-of-the-art distance measures is significantly competitive on actual data.

Furthermore, due to the fact that performing classification through the use of distance measures can be computationally expensive due to their quadratic complexity, we have explored the use of models, and specifically Hidden Markov Models (HMMs). HMMs are widely known and have been applied to a variety of domains, such as biology, speech recognition, and music retrieval, since they are able to model the underlying structure of sequences determining the relationships between their observations. Thus, in order to perform classification, we present a way of representing each class of a time series database with an HMM, and then perform searching based on the constructed models. The proposed indexing framework MTSI (shorthand for Model-based Time Series Indexing) works in a filter-and-refine manner, and is shown to be both effective and efficient compared to a variety of distance-based measures on a large number of time series databases. Additionally, we have exploited HMMs so as to extract knowledge on what has happened in the past or to recognize what is happening in the present. Specifically, instead of searching a database of time series by providing a time series corresponding, e.g., to a specific activity, for finding the time series that are similar to that query, we can provide a model representing this activity. Comparing HMMs and several distance measures for such searches shows that our approach is particularly meaningful with excellent accuracy results.

Finally, the problem of interval-based whole sequence matching has received limited attention, although it appears in many application domains, such as music informatics, sign language, medicine, geo-informatics, cognitive science, linguistics. Another contribution is a novel distance measure for event-interval sequences, called IBSM (abbreviation of Interval-Based Sequence Matching), which captures the similarity between event intervals, which are characterized by a label and a duration. Thorough experimental evaluation of IBSM on a variety of datasets demonstrates state-of-the-art performance.

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A problem of particular interest that has attracted the attention of music, database, and data mining communities during the last few decades is that of finding the best matching subsequence to a query in a large database. The problem of *subsequence matching* is defined as follows: given a query sequence and a database of sequences, identify the subsequence in the database that best matches the query. Achieving efficient subsequence matching is an important problem in domains where the target sequences are much longer than the queries, and where the best subsequence match for a query can start and end at any position in any sequence in the database.

A large number of Dynamic Programming (DP) [1] based distance or similarity measures perform similarity search in several application domains including time series, categorical sequences, and multimedia data. Nonetheless, there are still many application domains, such as music retrieval, where these methods are not directly applicable or have very poor retrieval accuracy, since in many cases several properties and characteristics of the specific domain are ignored. Thus, we focus on time series subsequence matching and approach the problem from the music retrieval perspective: suppose you hear a song but you cannot recall its name, or the melody of a song may sound similar to some other song that you cannot recall; in both cases one solution is to hum a short part of the song that you can easily remember and perform a search on a large music repository to find the song you are looking for or even songs with similar melody. The main task of a *Query-By-Humming* (*QBH*) system is, given a hummed query song, to search a music database for the K most similar songs. This

directly maps to subsequence matching as the hummed query is typically a very small part of the target sequence.

In order to guarantee robust and meaningful subsequence matching for a potentially very noisy domain, like QBH, several factors should be considered. First, when humming a song, errors may occur due to instant or temporary key or tempo loss, respectively. Thus, the matching method should be *error-tolerant*, otherwise there may be false negatives during the retrieval. In addition, the method should allow skipping a number of elements in both query and target sequences. Nonetheless, the number of allowed consecutive gaps in both query and target sequences should be bounded, in order to provide a setting that controls the expansion of the matched subsequences during the DP computation. Furthermore, to avoid producing very long matching subsequences, we should constrain the length of the matching subsequence, which can be set appropriately for the application domain. Also, to ensure that the matching subsequences will include as many matching elements as possible, the minimum number of matching elements may be lower bounded. However, constraining the number of matching elements may decrease the number of candidate matches. Specifically, if we have prior knowledge about the singing skills of the person who produced the hummed queries, we can tighten or loosen this constraint for strong or weak hummers, respectively. Finally, it is very important for a method to be used in QBH to choose a proper representation for music pieces that best fits their needs, without sacrificing efficiency and accuracy. QBH, along with music genre classification, are studied in Chapters 3, 4, and 5.

In this thesis, we also tackle the problem of *Nearest Neighbor* (*NN*) classification applied to *whole sequence matching*. First, given a query time series, a large database of time series, and a pool of distance measures, we are interested in identifying the most appropriate measure for the given query, so that we can perform NN

classification. In other words, we would like to annotate the query time series with the class of the closest database time series by applying a distance measure, which is the most suitable one out of several measures based on the specific query. This is a very tough problem, which, to the best of our knowledge, has not been addressed before. As shown in Chapter 6, the method we propose provides very competitive accuracy compared to the cross validation approach and individually using several distance measures. Secondly, we take advantage of the power of *Hidden Markov Models*, which are capable of modeling the underlying structure of sequences determining the relationships between their observations, so as to perform classification. Classes of time series are modeled via HMMs, and, given a query time series, we identify the models that have most likely produced that time series so as to narrow the search space. Then, the expensive distance-based computations take place between the query and the time series belonging to these classes. Again, the performance of our framework provides very competitive accuracy results, as shown in Chapter 7. In Chapter 8 we also demonstrate the usefulness of HMMs for searching time series databases, and compare our approach with several distance-based measures. Last but not least, we explore the NN classification problem on *event-interval sequences*. The main advantage of event-interval sequences over traditional event sequences, which model series of instantaneous events, is that they incorporate the notion of duration in their event representation. Essentially, sequences of event intervals can be encoded as a collection of labeled events accompanied by their start and end time values. Many application domains are characterized by sequences of event intervals, such as sign language [2, 3], medicine [4], geo-informatics [5], cognitive science [6], linguistics [7]. For example, in sign language, sentences are constructed by events corresponding to occurrences of various grammatical, syntactic, and gestural expressions. Moreover, in medicine [4], patients typically undergo a series of diagnostic tests and treatments

that have a time duration and may also occur concurrently. Melodies in music can also be represented by such sequences, where each note has a duration and a label, while it may also have multiple instances. In Chapter 9 we propose a method for *interval-based sequence matching*, which is shown to provide excellent results compared to the state-of-the-art methods on datasets of a variety of domains.

The main contributions of the thesis are the following:

- A subsequence matching framework that allows for a constrained number of gaps on both query and target sequences, variable tolerance levels in the matching (tolerances are functions of the query and target elements), and a matching range that constrains the maximum match length. Based on this framework, a similarity measure, SMBGT, is proposed, which given a query Q and a target sequence X finds the subsequence of X that best matches Q. Comparative evaluation on QBH of several DP-based methods, a probabilistic model-based method, and SMBGT on real and synthetic queries shows the superiority of SMBGT in terms of accuracy for several parameter settings.
- A embedding-based indexing approach, called ISMBGT, that works in a filter-and-refine manner and is designed for speeding up similarity search under SMBGT. Experimental evaluation of ISMBGT using synthetic and hummed queries on a large music database demonstrates that ISMBGT can achieve speedups of up to an order of magnitude against brute-force search under SMBGT, while maintaining a retrieval accuracy close to that of brute-force.
- An open-source QBH system named "Hum-a-song", which allows a user to hum a (part of a) song and search a large music repository to identify the top-K matches/songs. The user can select the method to be used from a pool of measures, provide the desirable music representation, and also the error tolerance to be applied.

- Exploiting SMBGT for music genre classification using the k-NN classifier.
- A framework for solving the problem of selecting the best measure, given a query and a pool of measures, for time series NN classification. The selection of the appropriate distance measure is performed via statistical significance testing on the pool of measures using a set of training time series. The framework is also query-sensitive. Within this framework three methods are employed that exploit different schemes to select the appropriate training time series for the statistical test. Experimental evaluation of the three methods against a baseline approach on a large collection of 45 time series datasets shows that our framework can achieve at least as good or better classification accuracies than the baseline on up to 35 datasets. Thus, it is highly competitive in terms of NN classification accuracy by using a pool of measures instead of using only one of them.
- A way of representing time series of a specific class via an HMM. Experimental evaluation of this representation against four distance measures in terms of classification accuracy on the training sets of 45 datasets demonstrates that HMMs can attain better or equal accuracy in 32 datasets. In addition, the performance of model-based search with HMMs against distance-based search with four measures on the test sets of the same datasets is examined. This evaluation indicates that the former type of search produces significantly better precision vs. recall tradeoffs than the competitors when the HMMs are trained with a sufficient number of time series.
- An indexing framework for time series NN classification, named MTSI. The framework works in a filter-and-refine manner, by exploiting the novel model-based representation of time series belonging to the same class. Extensive experiments on 33 datasets reveal that MTSI attains high accuracy as it is at least

as good as the baseline approach in 23 datasets, while achieving a speedup of up to an order of magnitude, showing both its effectiveness and efficiency.

- A novel method, called IBSM, for matching event-interval sequences that exploits a vector-based representation of event-interval sequences, which facilitates full sequence matching, by applying bilinear interpolation, and computes the Euclidean distance of the resulting sequence representations. Two techniques are also developed based on sampling and alphabet reduction that speed up the runtime and decrease the memory requirements of IBSM when performing NN search in a database of event-interval sequences. Experimental evaluation of IBSM against two state-of-the art measures on eight real datasets from different application domains shows that IBSM outperforms them in 7 out of 8 datasets in terms of NN classification accuracy, and by up to two orders of magnitude in terms of runtime.

CHAPTER 2

BACKGROUND

In this chapter, we first present some widely used terms in music, which will also be used in many chapters. Then, the representation methods for music pieces and time series will be given, and an overview of the matching methods that have been proposed, either for addressing QBH specifically or for other noisy sequence application domains that could be of interest in QBH as well, is provided. The discrimination of the different categories is done based on whether the methods perform sequence, model, or n-gram based similarity search [8].

2.1 Music Terminology

Next, we provide some basic terms used in music that will help the reader throughout the thesis. Every piece of music (as shown in Figure 2.1) is a sequence of notes characterized by a *key* that defines the standard pattern of allowed intervals that the sequence of notes should conform with, and a *tempo* that regulates the speed of the music piece. Each *note* consists of two parts, the *pitch* and the *duration*. A *pitch interval* is the difference in frequency of two adjacent notes. In western music the smallest pitch interval is called *semitone*, a *tone* comprises two consecutive semitones, and the interval of 12 consecutive semitones is called *octave*. Another important term is *transposition*, which is significant when representing music pieces as time series, and is defined as shifting a melody of a piece written in a specific key to another key. A time series can be seen as a variable evolving over time, sampled at regular time intervals. Finally, there is a discrimination between *monophonic* and

polyphonic music, where the latter allows two or more notes to sound simultaneously, as opposed to the former case. Figures 2.2 and 2.3 show a monophonic and its corresponding polyphonic part, respectively, of a well-known instrumental Classical piece of Wolfgang Amadeus Mozart, the "Rondo Alla Turca". Figure 2.4 presents the initial part of a polyphonic traditional orthodox church melody (with Greek lyrics) named "Here is darkness and morning" arranged for choir by S.N. Chatzistamatiou. We study monophonic music rather than polyphonic as in QBH we deal with melodies hummed by users. The interested reader can refer to the literature [9, 10, 11, 12, 13, 14] for approaches in polyphonic music.

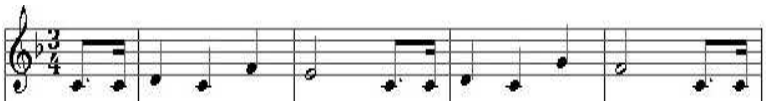


Figure 2.1: A part of the "Happy Birthday" music score.

It is important to understand the pros and cons of the different categories of methods (and certainly of the methods in particular), since music pieces are essentially sequences of values evolving in time, and thus the QBH problem is just an application accentuating the problematic aspects of methods that deal with time. With this in mind, improvements over existing methods can be introduced so that they can be used in other areas of interest, such as finding specific patterns in financial, weather, and sensor data observing the movements of humans. In the latter case assistive actions can be taken when particular patterns are observed.

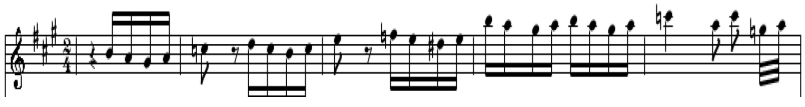


Figure 2.2: Initial part of the "Rondo Alla Turca" monophonic music score.

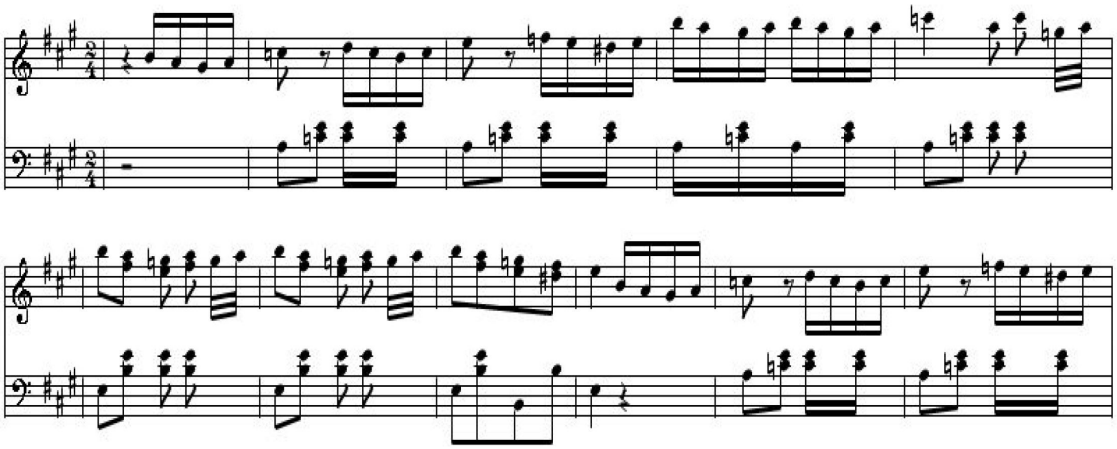


Figure 2.3: Initial part of the "Rondo Alla Turca" polyphonic music score.

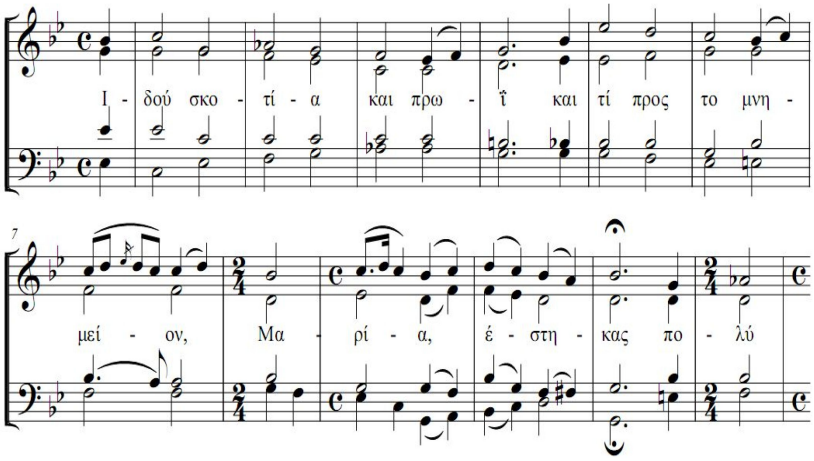


Figure 2.4: Initial part of the "Here is darkness and morning" polyphonic music score (with Greek lyrics).

Next, we present different ways of representing music pieces and time series, along with the variety of matching methods that are, or may be, of interest in QBH and other similar noisy sequence domains, categorized in sequence, model, and n-gram based methods.

2.2 Representing Music Pieces

Songs are essentially music audio signals that can be either represented in symbolic format based on musical scores, or in audio format based on analogue signals, which can be sampled and encoded in different ways. A widely used format of the first type of representation is MIDI [1], while Humdrum has also been used. Although the MIDI has some disadvantages, such as that the sound quality depends on synthesizer and it cannot store voice, it has some important advantages. It takes very little space making it easy to store and communicate, and it provides a format that can be easily handled allowing easier comparison between different instruments and music pieces. Thus, it has been widely accepted, and in the next chapters we focus on this type of symbolic format.

There are three common ways to express pitch: (a) *absolute pitch*, where the frequency of the note is used [15, 16, 17, 18] - in MIDI, the music format we are interested in, this value is an integer in $[1, 127]$ (0 corresponds to *pause*) - (b) the note *number in the key* of the piece [19], and (c) *pitch interval*, the frequency difference between two adjacent notes. We note that for the second way of representing pitch the key does not usually remain the same throughout the whole music piece, which makes this encoding very hard to use. Pitch intervals can be transformed/quantized in [-11,11] by applying modulo 12 [20], leading to two octaves, a reasonable range as the human singing range rarely goes beyond this interval.

Regarding the encoding of duration there are three options [21]: (a) *Inter-Onset-Interval* (IOI), defined as the difference in time onsets/clicks of two adjacent notes, (b) *IOI Ratio* (IOIR), defined as the ratio of IOIs of two adjacent notes with the IOIR of the last note equal to 1, and (c) *Log IOI Ratio* (LogIOIR), the logarithm

[1] Musical Instrument Digital Interface

of IOIR. A variation of LogIOIR is to quantize the LogIOIR values of the notes to the closest integer or the closest value in $[-2, 2]$ [21].

Although many approaches represent notes by encoding only pitch [19], it has been shown that the use of both pitch and duration information improves music retrieval [22] and the melody sequences are made more unique [17, 20]. For example, two or more songs may share similar note frequencies (i.e., pitch values) but their melodies may vary due to different individual pitch durations. Hence, if, for instance, only pitch is used to represent a music song, there is a high risk of erroneously matching two songs. As a result, we take into account both pitch and duration information. Melodies are represented by 2-dimensional time series of notes of arbitrary length, where the first dimension corresponds to pitch and the second one to duration.

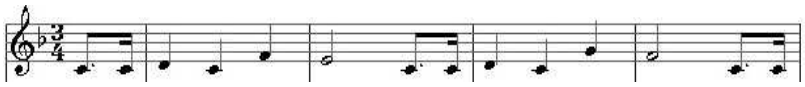
Considering all possible combinations of pitch and duration encoding, we can easily conclude that pitch interval and IOIR (or variations of them) leads to transposition and time invariance when comparing melodies. Thus, with these combinations we deal with note transitions, saving much computational time as we do not have to check for possible transpositions of a melody, nor do we have to scale in time when comparing melodies. Such combinations are shown in Table 2.1, and any of them can be incorporated in the 2-dimensional representation of music pieces. In Figure 2.5(b) we can see an example of the $\langle$pitch interval, IOIR$\rangle$ representation of the part of "Happy Birthday" song shown in Figure 2.5(a).

2.3 Time Series Representations

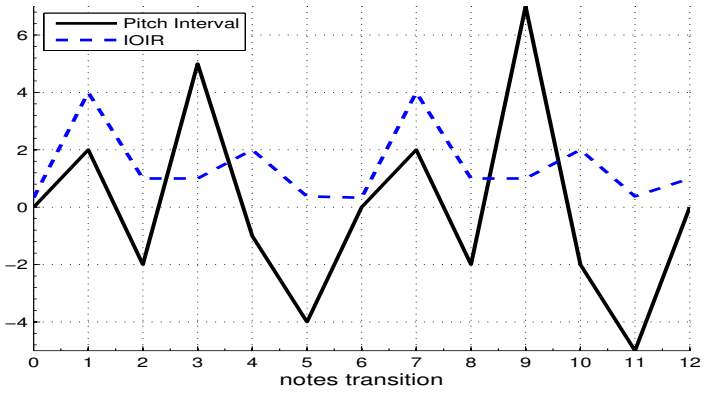
One recent line of research has been focusing on identifying discriminant time series subsequence patterns, known as Shapelets [23] and variants [24, 25, 26, 27]. Shapelets are mainly used for time series classification, since due to their construc-

Table 2.1: Code numbers for 2-dimensional representations

Code number	Representation
1	⟨mod12, IOIR⟩
2	⟨mod12, LogIOIR⟩
3	⟨mod12, LogIOIR in $[-2, 2]$⟩)
4	⟨mod12, LogIOIR quantized to closest integer⟩
5	⟨pitch interval, IOIR⟩
6	⟨pitch interval, LogIOIR⟩
7	⟨pitch interval, LogIOIR in $[-2, 2]$⟩)
8	⟨pitch interval, IOIR quantized to closest integer⟩



(a) Part of the music score.



(b) Representation using pitch intervals and IOIR.

Figure 2.5: Example of the music score and its 2-dimensional time series representation. IOIR is the duration ratio of two consecutive notes.

tion they are expected to be more informative and representative of some class. Other time series representations that capture global or local structural characteristics include SpaDe [28], DFT [29], SAX [30], and Bag-Of-Patterns [31]. For a comparison of a variety of representations, please refer to Wang et al. [32], which demonstrates that there is little difference among them. Speeding up similarity matching in large databases based on representations has also attracted the attention of researchers. F-Index [29] exploits DFT to build an index on which a lower bound of the Eu-

clidean distance is applied, before the expensive computation in the original space is performed.

2.4 Sequence Matching

There are two broad categories of matching between sequences, the *whole/full sequence matching* and the *subsequence matching*. Next, we give a brief overview of methods of these categories. It should be mentioned that some of them have already been applied to music retrieval, while others not. It is of particular importance to identify the pros and cons of such methods, so that we can further decide if some of these methods can be improved to apply to QBH where the music pieces are represented as sequences of notes.

2.4.1 Whole sequence matching

Referring to the first category of whole sequence matching, the most common measure for computing the distance between time series is the dynamic programming (DP) method [1] Dynamic Time Warping (DTW) [33]. DTW is a distance measure allowing to compare two sequences that may vary in time or speed. Since DTW has been highly appropriate for time series similarity measurement, several lower bounds have been proposed to speed up its expensive computation [34, 35, 36]. There are also several variants of DTW, among which constrained DTW (cDTW) [37, 38], Edit Distance on Real sequence (EDR) [39], and Edit distance with Real Penalty (ERP) [40] widely used in many application domains. The most attractive property of these algorithms is that they are robust to misalignments along the temporal axis, i.e., to differences in the speed in which observations evolve across time. Some of these methods, such as DTW and cDTW, have been shown to achieve high accuracy in applications such as time series mining and classification [32, 39, 41]. The first

variant, cDTW, avoids the matching of elements that are far away from each other by bounding the difference of the sequences' indices. EDR [39] and ERP [40] are DP-based distance measures and use the triangle inequality for pruning purposes; the latter is also a metric. It should be noted that ERP is more robust to noisy data. A novel measure for time series, called Move-Split-Merge (MSM), was proposed very recently [42]. MSM is metric and uses three fundamental operations, Move, Split, and Merge, which can be applied in some sequence to transform any time series into any other time series. In addition, it is invariant to the choice of origin, as opposed to ERP. The Edit distance [43], which is a metric measure, can be used to find the distance between two strings. It is defined as the minimum number of edit operations needed to transform one string into the other, with the allowable operations being insertion, deletion, and substitution of a single character. Last but not least, the Time Warp Edit Distance (TWED) [44] is a metric distance measure whose goal is to find a sequence of edit operations (deleting an element from any time series and match two elements) allowing for the simultaneous transformation of two time series so as to superimpose them with minimal cost. Additionally, it is an elastic measure supporting local time shifting using timestamp differences between compared points. The method proposed by Adams et al. [45] operates using DTW and has been applied to music retrieval. It takes into account absolute pitches and deals with the problems of invariance by applying occasionally normalizations, depending on the proposed representation. Another approach that incorporates tempo variations was introduced by Mongeau and Sankoff [19], while Dannenberg and Hu [46] modified the latter approach to deal with various representations as well. To deal with tempo variation the method presented by Mazzoni and Dannenberg [47] scales the target sequences before applying DTW.

Apart from these DTW-based whole sequence matching methods, the similarity between two sequences can be measured by finding their Longest Common SubSequence (LCSS) [20, 48, 49, 50], i.e., the greatest number of elements that are common in both of them, and this can be done again using a DP approach. Thus, LCSS-based approaches can tolerate noise introduced by the user in the QBH application, while they allow for gaps on both sequences during their alignment. However, since no bounds are imposed to the allowed gaps this may result in a large number of false positives when $|Q| << |X|$. Since errors in QBH systems are imposed by the users, tolerance in pitch and duration is highly eligible. Though, we should not be tolerant to skip too many elements in the target sequence, as in that case there is small possibility of finding a good match. With this direction in mind, Iliopoulos and Kurokawa [51] proposed an algorithm which accounts for a bounded number of gaps only in the target sequence. The algorithm also handles errors by using constant values as tolerances for each query element. Variations regarding the sum, differences, and length of consecutive gaps are proposed by Crochemore et al. [52] when matching two sequences, but, as in the work by Iliopoulos et al. [51], they are allowed to exist in the target sequence only. Notice that both approaches deal with *whole query matching* and absolute pitches, while none of them accounts for note duration, i.e., they are proposed for 1-dimensional sequences. Some approaches embed transposition invariance as a cost function in the DP computation [53, 54], though, with not attractive runtime. In addition, Lemstrom et al. [54] and Deorowicz [55] aim at matching the whole query ignoring duration. Transposition invariance can also be encoded by a proper representation [56]. Other proposed distance/similarity measures are TQuEST [57], DISSIM [58], and CID [59].

2.4.2 Subsequence matching

To efficiently address the QBH application, where it is more suitable to use subsequence matching methods, the aforementioned whole sequence matching methods could be retrofitted by performing a sliding window search over the database. The issue with such an approach is that it would be computationally expensive requiring one DP computation per window. The approach of Han et al. [60] is based on uniform segmentation and sliding windows, which requires the user to manually select the length of the segments and is therefore not sensitive to the actual behavior of the data and can efficiently handle only *near exact* matching.

Some DP methods reduce subsequence matching to full sequence matching, by cutting database sequences into small pieces, and requiring each query to correspond to an entire such piece. Such approaches though fail when the query corresponds to a database subsequence that is not stored as a single piece. One example is the Query-By-Humming system described by Zhu et al. [61], where each database song is cut into smaller, disjoint pieces, and they developed an efficient lower-bound for DTW. As a result, this method is not directly applicable to subsequence matching.

Two similar methods for subsequence matching are presented by Hu et al. and Jang et al. [62, 63], which implicitly account for tempo scaling. However, they are not transposition invariant, as absolute pitches are used. Furthermore, the Edit distance [43] has been used for music retrieval with several variations [53, 54, 64, 65]. Its most recent version used in QBH is presented by Unal et al. [66], where the cost function has been properly modified so that this distance measure uses both pitch and duration information. Another DP-based method for finding the subsequences of evolving numerical streams that are closest to a query is presented by Sakurai et al. [67]. SPRING is based on DTW and does not sacrifice accuracy, in spite of

the fact that DTW is not a metric distance function. Another reason for making this method attractive is that it operates in constant space, and time linear to the database size (for short queries). Thus, it is a promising method to be applied to the problem of QBH. A method for improving the efficiency of subsequence matching under unconstrained DTW is described in Zhou and Wong [68], where it is assumed that the length of the optimal subsequence is known, and equal to the length of the query. Its best-case complexity is $O(mn)$, where m is the size of the query and n is the size of the long sequence that we search for subsequence matches. We note here that, unlike the method proposed by Zhou and Wong [68], SPRING does not place any constraint on the length of the subsequence match. Smith-Waterman [69] is a local alignment method for finding similarity, which compares segments of all possible lengths optimizing the similarity measure, and has been applied to QBH [20].

2.5 Model-Based Matching

Several probabilistic methods (HMM-based) have been developed for speech recognition and music retrieval [66, 70, 71, 72, 73, 74]. An extended HMM is used by Meek et al. [71], where the target and query notes are associated through a series of hidden states, modeling the local and cumulative error in pitch and durations. HMMs have also been used to compute the match score of the local alignment when both pitch and duration [72] are exploited. An extended HMM architecture Factorial HMM has been proposed to model music, and more specifically Bach's chorales [70]. Factorial HMMs are based on a factored, distributed representation of the hidden state variable. Due to its complicated structure, inference and learning is intractable, and approximate learning is necessary. Although the model may be effective in capturing the statistical structure in the Bach's chorales, it is not built for any query processing

as QBH. Unal et al. used HMMs for the segmentation of the humming instances, followed by energy and pitch analysis to correct the segmentation errors. They use both pitch and duration and the retrieval stage is done by using salient information in the transcribed symbolic sequences (FingerPrints), but their method does not significantly outperform edit distance [66]. Although the HMM-based methods are suitable for modeling the probabilistic behavior that is inherent in an application domain such as music retrieval, they are computationally expensive due to the required training, and creating models to represent and discriminate all the different genres of music in a large database is a very tough task. Other approaches that exploit HMMs are presented by Shih et al. [75, 76], though their focus is to use HMMs as their front end to capture the location of notes in the input and not perform retrieval.

2.6 n-grams

A method that has been successfully used in string matching is *n-grams*, according to which the number of matching substrings of a fixed length n is counted. Also, since it is more likely for long sequences to have a match, this count should be normalized. An efficient method for approximate string matching is proposed by Ukkonen [77], but is not suitable for the case of QBH where we have to deal with very noisy queries. n-gram-based methods proposed for music retrieval [20, 78] fail to handle noisy queries efficiently leading to poor performance as they are designed for *near exact* matching, although they may keep pitch and duration information.

2.7 Open Problems in QBH

Having studied the aforementioned methods for solving the similarity search problem in QBH, we observed several issues that need to be taken into account when

trying to address this problem. First, there is no standard and preferred music format to store music pieces. MIDI and MP3 are the mostly used formats for this problem, and although MP3 is richer in information, it cannot be easily extracted and exploited as opposed to MIDI. Second, there is no music database widely available that can be used as a testbed to test the proposed methods on, and as a result the results reported are most of the times subjective, or at least cannot be generalized since the datasets are quite small and limited to a few genres of music or to songs of Beatles. Moreover, to the best of our knowledge, automatic music genre classification, which would help efficiency and accuracy, is not part of the proposed end-to-end systems. Last but not least, since QBH is a very noisy application domain, a method should allow for both slight and more serious key or tempo loss errors at the same time without affecting accuracy.

2.8 Conclusions

In this chapter, we first presented a brief overview of the ways to encode music pieces and time series. Also, we have presented methods for matching sequences that have been proposed in the literature for the QBH application or that would be of interest for the community to study in more detail. The latter would help in better understanding the problematic aspects of applying them to similar application domains.

CHAPTER 3

A SUBSEQUENCE MATCHING WITH GAPS-RANGE-TOLERANCES FRAMEWORK

3.1 Introduction

In this chapter, we first describe the $SMBGT$ (shorthand for Subsequence Matching with Bounded Gaps and Tolerances) method that is based on a framework, which, given a query Q and a target sequence X (with $|Q| << |X|$), finds the subsequence of X that best matches Q [79]. SMBGT is error-tolerant and allows gaps in the alignment, which and can be bounded in both query and target sequences (by β and α respectively). Moreover, the maximum match length in X as well as the minimum number of matching elements may be constrained (by r and δ respectively). An example of SMBGT is shown in Figure 3.1.

However, one limitation of SMBGT is its computational time complexity, which is $O(|Q||X|)$ [79]. For application domains with many and large database sequences, such as QBH, the runtime is critical. Hence, a method for efficient similarity search under SMBGT is needed, that can achieve significant speedups against brute-force search with minor losses in accuracy. As a result, a novel embedding-based filter-and-refine approach is also presented, which we call *ISMBGT*, shorthand for Indexed Subsequence Matching with Bounded Gaps and Tolerances. ISMBGT is designed to improve the efficiency of processing subsequence matching queries in time series databases under the SMBGT method. The key idea is that the subsequence matching problem can be partially converted to the much more manageable problem of nearest neighbor retrieval in a real-valued vector space. This conversion is achieved by

defining an embedding function that maps each database sequence into a sequence of vectors, which we call database sequence embeddings. There is a one-to-one correspondence between each such vector and a position in the database sequence. The embedding function also maps each query sequence into a vector, which we call query embedding. The mapping is performed in such a way that if the query is very similar to a subsequence of the database, the new vector-based representation of the query is likely to be similar to the vector corresponding to the endpoint of that subsequence.

These vectors are computed by matching queries and database sequences with the so-called reference sequences, that is, a relatively small number of preselected sequences. The expensive operation of matching database and reference sequences is performed offline. At runtime, the query time series is mapped to a vector by matching the query with the reference sequences, which is typically orders of magnitude faster than matching the query with all database sequences. Then, promising candidates for the best subsequence match are identified by finding the nearest neighbors of the query vector among the database vectors. Applying sampling on the database vectors demonstrates that this process can be accomplished even faster. An additional refinement step is finally performed, where subsequences corresponding to the top vector-based matches are evaluated using the SMBGT method.

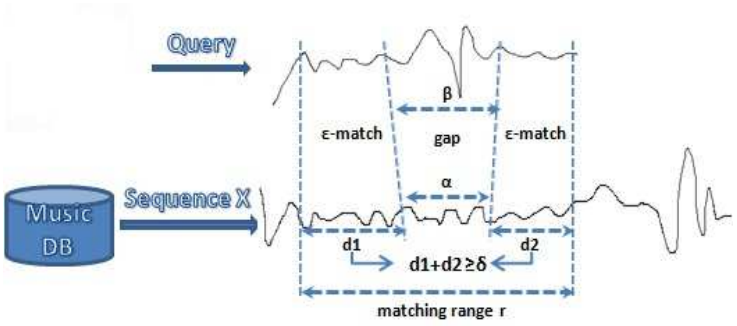


Figure 3.1: SMBGT: error-tolerant matching is denoted as ϵ-match.

We note that ISMBGT differs substantially from existing embedding-based indexing methods, in that

- it does not require any training, which can be prohibitively costly for large databases, for creating the embeddings of the database sequences: reference sequences are selected randomly from the database and cover all ranges of query lengths,
- it is query-sensitive: the construction of the query vector is performed using a technique that optimizes the selection of the reference sequences based on the query length,
- it exploits the flexibility of SMBGT: it allows for a bounded number of gaps in both the query and target sequences, variable tolerance in the matching, a matching range in the alignment, and also optionally constrains the minimum number of matching elements.

3.2 Related Work

Indexing methods for sequence matching can be divided based on the underlying distance or similarity measure that they target. In string matching, OASIS [80] employs a best-first search technique over a suffix tree for string alignment under the Smith-Waterman similarity measure, and achieves significant speedups over the brute-force application of Smith-Waterman. However, this method is not directly applicable to the problem of subsequence matching of time series, which typically uses values obtained from a continuous space, such as the space of real numbers or higher-dimensional vector spaces. One way of retrofitting these methods for time series subsequence matching is to transform both the time series database and the query into strings [81, 82] and use these methods directly for retrieving the best match.

Such an approach, however, will produce results different from those given by specific subsequence matching methods (e.g., under DTW). Moreover, the limited number of symbols in strings has been exploited to design indexing methods based on q-grams [82, 83] and suffix trees [80, 84], while other embedding-based methods have been developed for subsequence matching in large string databases [85]. Applying such methods to our problem setting requires discretizing time series (i.e., real values), which is not a very trivial task.

An indexing structure for unconstrained DTW-based subsequence matching has been proposed [86]. However, as database sequences get longer, the time complexity for that method becomes similar to that of unoptimized DTW-based matching. In contrast, the method in Park et al. [87] can handle such long database sequences by using monotonicity, but is only applicable to 1-dimensional time series. A related method that can be used for multidimensional time series is named piece-wise approximation method (PAA) [88], where time series are approximated by shorter sequences obtained by replacing each constant-length part of the original sequence with the average value over that part. The lower-bounding method LB_Keogh for time series matching is described in Keogh et al. [41]. The main idea is to use the warping constraint to create an envelope around the query sequence. Then, using a sliding window of size equal to the query, a lower bound of the matching cost between the query and each possible subsequence can be estimated, and thus it can be used to prune a large number of subsequences. Nonetheless, its performance deteriorates as warping width and query size increase, and its computation for each possible subsequence can be time consuming for large databases. Proposed improvements to the LB_Keogh (e.g., [89]) achieve not more than a small constant factor in terms of both the tightness of the lower bound and the query time performance. The DTK method [90] is a method for subsequence matching under cDTW. DTK breaks the database

into small non-overlapping sequences and further employs PAA [88] for efficient indexing. This approach, however, does not scale well as the query size increases. A similar approach is used to index time series for sequence and subsequence matching under scaling and DTW [91].

The indexing approach proposed is embedding-based. Several embedding methods exist in the literature for speeding up distance computations and nearest neighbor retrieval. Examples of such methods include Lipschitz embeddings [92], FastMap [93], MetricMap [94], SparseMap [95], and BoostMap [96, 97]. Such embeddings can be used for speeding up full sequence matching [95, 96, 97]. Similar embedding-based methods have been developed for subsequence matching in time series (EBSM [98]) and string (RBSA [85]) databases. In this chapter, we present an indexing method for SMBGT that achieves significant speedups against brute-force search. We chose SMBGT because it is very competitive in terms of retrieval accuracy against existing DP-based and HMM-based methods for QBH, as shown in the experiment evaluation.

3.3 Problem Setting

3.3.1 Definitions

Let us now give a more general problem setting. Consider $X = (x_1, \ldots, x_{|X|})$ to be a multi-dimensional time series, where $|X|$ denotes the length of X. We use x_j^d to denote the d-th dimension of x_j, where $j = 1, \ldots, |X|$. In our case, where X represents a music piece, each $x_j = \langle x_j^1, x_j^2 \rangle \in X$ is a pair of real values, where x_j^1 and x_j^2 correspond to pitch and duration information respectively, and are represented using any of the schemes described in Section 2.2. A music database is a set of time series $DB = \{X_1, \ldots, X_N\}$, where N is the number of music pieces in DB. A *subsequence* of X, denoted as $X[ts : te] = \{x_{ts}, \ldots, x_{te}\}$, is an ordered set of elements

from X appearing in the same order as in X. The first element of the subsequence is x_{ts} and the last is x_{te}. Note that $X[ts:te]$ is not necessarily continuous, i.e., gaps are allowed to occur by skipping elements of X. Let $Q = (q_1, \ldots, q_m)$ be another multi-dimensional time series, and consider the following definitions:

Definition 1. (*Variable error-tolerant match*) Consider elements $q_i \in Q$ and $x_j \in X$. For each dimension d of x_j and q_i, we define a function $\epsilon_d^f(i,j) = f(q_i^d, x_j^d)$. We say that q_i and x_j *match with variable ϵ-tolerance*, and denote it as $q_i \approx_\epsilon^f x_j$, if for each d a constraint $\mathcal{C}(q_i^d, x_j^d, \epsilon_d^f)$ is satisfied.

Depending on the form of $\mathcal{C}$ we can define two types of variable tolerance, i.e., absolute and relative:

- *variable absolute error tolerance*: values q_i^d, x_j^d may differ by at most ϵ_d^f:

$$\mathcal{C}(q_i^d, x_j^d, \epsilon_d^f) = \{|x_j^d - q_i^d| \leq \epsilon_d^f\}. \tag{3.1}$$

- *variable relative error tolerance*: value q_i^d may vary between a fraction and a multiple of value x_j^d:

$$\mathcal{C}(q_i^d, x_j^d, \epsilon_d^f) = \{x_j^d/(1+\epsilon_d^f) \leq q_i^d \leq x_j^d * (1+\epsilon_d^f)\}. \tag{3.2}$$

Note that if $\epsilon_d^f(i,j)$ is a constant function then the last two tolerances are called *constant* absolute and relative error tolerances, respectively.

Based on this definition, it can be observed that the proposed variable tolerance framework is generic and can be used for any time series application domain. In Section 3.3.1.1, we present an instantiation of the above definition for the 2-dimensional time series used in the QBH application.

Definition 2. (*Common bounded-gapped subsequence*) Consider two subsequences $Q[ts_1{:}te_1]$ and $X[ts_2{:}te_2]$ of equal length. We use $\mathcal{G}_Q$ and $\mathcal{G}_X$ to denote the indices of those elements in Q and X, respectively, that are included in the corresponding subse-

quences. Let α and β be the number of consecutive elements that can be skipped in X and Q, respectively. If $q_{\pi_i} \approx_\epsilon^f x_{\gamma_i}$, $\forall \pi_i \in \mathcal{G}_Q$, $\forall \gamma_i \in \mathcal{G}_X$, $i = 1, \ldots, |\mathcal{G}_Q|$, and

$$\pi_{i+1} - \pi_i - 1 \leq \beta,\ \gamma_{i+1} - \gamma_i - 1 \leq \alpha, \tag{3.3}$$

then, pair $\{Q[ts_1{:}te_1], X[ts_2{:}te_2]\}$ defines a *common bounded-gapped subsequence* of Q and X. The longest such subsequence satisfying $te_2 - ts_2 \leq r$ is called *Longest Common Bounded-Gapped Subsequence* and denoted as $LCBGS(Q,X)$.

3.3.1.1 Variable Tolerances - Instantiation

A reasonable definition for variable ϵ_1^f (where $d = 1$ corresponds to the pitch dimension) is the following:

$$\epsilon_1^f(i,j) = f(q_i^1) = \lceil q_i^1 * t \rceil,\ \text{with } t = 0.2,\ 0.25,\ 0.5. \tag{3.4}$$

Note that ϵ_1^f is a function of only the first dimension of q_i, i.e., the pitch.

Using ϵ_1^f given by Equation 3.4, the variable relative error tolerance for pitch is defined as follows:

$$\mathcal{C}(q_i^1, x_j^1, \epsilon_1^f) = \begin{cases} x_j^1/(1+\epsilon_1^f) \leq q_i^1 \leq x_j^1 * (1+\epsilon_1^f), q_i^1, x_j^1 \geq 0, \\ x_j^1/(1+\epsilon_1^f) \geq q_i^1 \geq x_j^1 * (1+\epsilon_1^f), q_i^1, x_j^1 < 0. \end{cases} \tag{3.5}$$

Note that in the equation above it was necessary for the given application to distinguish between positive and negative values of pitch intervals.

For pitch, in our evaluation we experimented with constant and variable absolute and relative tolerances.

For duration, we had to differentiate between the two representations used in this framework, IOIR and LogIOIR. Hence, for finding a suitable form for $\mathcal{C}$ we asked people to hum several pieces of different kinds of music and we observed a tendency

of making duration ratios smaller, even half of their actual values. This is reasonable, as users care more about singing melodies than being exact in tempo. Also, our definition of $\mathcal{C}$ should account for cases of queries at slower tempos.

Thus, for IOIR, we define the following form of variable error tolerance without providing any function ϵ_2^f:

$$\mathcal{C}(q_i^2, x_j^2, -) = \{x_j^2 \leq 2 * q_i^2 \text{ and } x_j^2 - q_i^2 \geq -0.5\}. \tag{3.6}$$

Furthermore, for the case of LogIOIR, taking into account that negative values may occur, we define the following form of variable error tolerance, again without providing any function ϵ_2^f:

$$\mathcal{C}(q_i^2, x_j^2, -) = \begin{cases} \{0 \leq log_2(x_j^2/q_i^2) \leq 1\}, log_2 x_j^2 \geq 0. \\ \{|log_2(x_j^2/q_i^2)| \leq 1\}, log_2 x_j^2 < 0. \end{cases} \tag{3.7}$$

As the logarithmic values get smaller, the difference in ratios gets smaller as well. Notice that the two forms of $\mathcal{C}$ shown in Equation 3.6 and Equation 3.7 are appropriate for variable error tolerance for the QBH application.

Example: Let $Q = (6, 3, 10, 5, 3, 2, 9)$, $X = (1, 1, 3, 4, 6, 9, 2, 3, 1)$. For simplicity we assume that Q and X are 1-dimensional. Now, consider subsequence Q[2:6] with $\mathcal{G}_Q = \{2, 4, 5, 6\}$, which corresponds to sequence $\{3, 5, 3, 2\}$, and X[3:8] with $\mathcal{G}_X = \{3, 4, 7, 8\}$, which corresponds to $\{3, 4, 2, 3\}$. Also, assume the following parameter setting: $\epsilon_1^f = 1$ (constant absolute tolerance) and $\epsilon_2^f = 0$ (since we only consider 1-dimensional time series), $\alpha = 2$, $\beta = 1$, and $r = 6$. Clearly, the two subsequences are of the same length, at most two ($\alpha = 2$) consecutive gaps occur in X (between the second and third elements in X[3:8]), and at most one ($\beta = 1$) consecutive gap occurs in Q (between the first and second elements in Q[2:6]). Range constraint $r = 6$ clearly holds for X[3:8], while all matching elements in the two subsequences differ by at most 1 ($\epsilon_1^f = 1$). Thus, pair $\{Q[2:6], X[3:8]\}$, is the $LCBGS(Q,X)$.

3.3.2 Problem Formulation

Problem: (*Top-K Subsequence Matching*) Given a database DB with N sequences of arbitrary lengths, a query sequence Q, and positive integers δ and r, find set $\mathcal{S} = \{X_i[ts : te] | X_i \in DB\}$ with the K subsequences having the largest $LCBGS$, such that

$$|LCBGS(Q, X_i[ts : te])| \geq \delta. \tag{3.8}$$

It should be mentioned that each database sequence contributes with only one subsequence $X_i[ts : te]$ to $\mathcal{S}$. Note the additional constraint $te - ts \leq r$ which is by definition included in LCBGS.

3.4 Background Methods

In this section, we briefly describe five DP-based methods and a probabilistic method that are promising for QBH.

DP-based methods typically use an "alignment array" a of size $(|Q|+1)*(|X|+1)$, where Q, X are the compared sequences. For $1 \leq i \leq |Q|$ and $1 \leq j \leq |X|$, each cell $a_{i,j}$ represents either the minimum cost for aligning subsequences of Q and X ending at i and j, respectively, or the maximum number of their matched elements, depending on whether the method uses a distance or similarity measure. We have to note that all methods given below except for the Iliopoulos et al. and the probabilistic method are designed for subsequence matching.

3.4.1 Edit distance-based

The most recent variation of the *Edit distance* [66] between two sequences Q and X, with slight extensions to deal with LogIOIR and quantizations, is computed as follows:

$$\begin{aligned} a_{0,j} &= 0 \text{ and } a_{i,0} = i, \\ a_{i,j} &= \min\left\{a_{i-1,j}+1, a_{i,j-1}+1, a_{i-1,j-1}+w(q_i,x_j)\right\}, \end{aligned} \tag{3.9}$$

where $w(q_i, x_j)$ is defined as:

$$w(q_i,x_j) = \frac{1}{2} * \left\{ \left| \frac{q_i^1 - x_j^1}{PitchRange} \right| \right\} + \frac{1}{2} * DurationCost \tag{3.10}$$

$$DurationCost = \begin{cases} |1 - \frac{\min\left\{q_i^2, x_j^2\right\}}{\max\left\{q_i^2, x_j^2\right\}}|, \text{ for IOIR.} \\ |\frac{q_i^2 - x_j^2}{DurationRange}|, \text{ for LogIOIR.} \end{cases} \tag{3.11}$$

PitchRange and *DurationRange* correspond, respectively, to the maximum pitch interval and LogIOIR range in DB. After a is computed, this method reports $\min_j\{a_{|Q|,j}\}$, i.e., the minimum cost of aligning Q with the subsequence of X ending at position j.

3.4.2 SPRING

Apart from the alignment array a, *SPRING* [67] also uses an additional matrix s, which keeps for each cell $a_{i,j}$ the start point of its current best alignment. The recursive computation of $a_{i,j}$ is:

$$a_{0,j} = 0 \text{ and } a_{i,0} = \infty, \tag{3.12}$$

$$a_{i,j} = w(q_i,x_j) + d_{best}, \tag{3.13}$$

$$d_{best} = \min\left\{a_{i-1,j}, a_{i,j-1}, a_{i-1,j-1}\right\}, \tag{3.14}$$

with $w(q_i, x_j)$ being the L_p norm of q_i and x_j. The same initialization is used for $s_{i,j}$ and at each iteration the start point of the element that was used to produce d_{best} is propagated. Finally, after a is computed, SPRING reports $\min_j\{a_{|Q|,j}\}$.

3.4.3 DTW tempo scaling

The next two methods were developed to measure the melodic similarity of two sequences Q and X without using tempo information directly, though allowing to locally adjust the tempo at certain positions. We refer to them as *simple* (DTW_s) [63] and *complex* (DTW_c) [62]. Both methods share the same initial condition, shown in Equation 3.15, where the simple scheme has been slightly modified to conform with the complex scheme. The recursions for the simple and complex scaling schemes are shown in Equation 3.16 and Equation 3.17, respectively.

$$a_{0,j} = 0 \text{ and } a_{i,0} = a_{i-1,0} + c, \tag{3.15}$$

$$a_{i,j} = w(q_i^1, x_j^1) + \min\left\{a_{i-1,j-1}, a_{i-1,j-2}, a_{i-2,j-1}\right\}, \tag{3.16}$$

$$a_{i,j} = \min\left\{\begin{array}{l} a_{i-1,j-1} \\ a_{i-2,j-1} + w(q_{i-1}^1, x_j^1) \\ a_{i-1,j-2} + w(q_i^1, x_{j-1}^1) \end{array}\right\} + w(q_i^1, x_j^1). \tag{3.17}$$

Note that c is a user-defined positive integer and $w(q_i^1, x_j^1) = |q_i^1 - x_j^1|$. Finally, both schemes report $\min_j\{a_{|Q|,j}\}$.

We note that the time complexity of the four aforementioned methods is $O(|Q||X|)$.

3.4.4 Iliopoulos et al.

The method proposed by *Iliopoulos et al.* [51], named here *Il. et al.* in short, performs *whole query* matching, by demanding all elements of Q to match within a constant ϵ-tolerance in a subsequence of X, and allows for a limited number of gaps

only in the target sequence. A DP computation is performed, where every match is rewarded with a score of 1, whereas whenever a mismatch occurs between q_i and x_j it checks whether the best matching value found so far for q_i can be propagated without exceeding α gaps in X. The time complexity of this method is $O(|X| + |Q||X|^2)$.

3.4.5 Hidden Markov Model-based Method

Since probabilistic methods have also been applied to music retrieval, for completeness, in the first part of our experiments we evaluate all the aforementioned methods with a *Hidden Markov Model* (*HMM*)-based method.

An HMM [73, 99] is a doubly stochastic process that contains a finite set of states. Each state emits/observes one symbol based on a probability distribution, and transitions between states are regulated by the so-called *transition probabilities*. More formally, an HMM is defined by: (1) M distinct states, (2) L distinct symbols that can be observed at each state, i.e., the discrete alphabet [1], (3) set $T = \{t_{ij}\}$ of transition probabilities, where $t_{ij} = P[s_t = j | s_{t-1} = i], 1 \leq i, j \leq M$, where s_t is the state at time t (first order Markov chain assumption), (4) set $E = \{e_j(k)\}$ of the probabilities of observation symbols at state j, where $e_j(k) = P[o_t = k | s_t = j]$ and o_t is the observed symbol at time t, and (5) set $\Pi = \{\pi_j\}$ of prior probabilities, where $\pi_j = P[s_1 = j], 1 \leq j \leq M$.

Given a database of sequences, if we had a probabilistic model for each individual sequence or group of homogeneous sequences, we could transform the query matching problem to a probability calculation of each model having generated a sequence (query Q). In other words, we would be looking for the model that maximizes the log-likelihood of the query sequence.

[1] Depending on the time series domain, real numbers can also be observed.

In QBH the database may contain a large number of songs covering a wide range of music genres, as happens with our data. This would impose high heterogeneity in the database and there would be no implication about any kind of correlation between the sequences. Thus, it is obvious that the most reasonable and fairest approach would be to model each database sequence with one HMM [72], which is in fact the approach we adopted. Regarding the time complexity of this method, training an HMM for a sequence X is $O(W|X|M^2)$, and computing the log-likelihood of a query Q being generated by an HMM is $O(|Q|M^2)$, where W is the number of iterations of the Baum-Welch algorithm [73].

3.5 SMBGT

In this section, we first present a method, called $SMBGT$, for identifying the LCBGS of a query Q and a sequence X [79]. We also demonstrate how the method operates through an example.

3.5.1 Method

SMBGT bounds the number of consecutive gaps allowed in *both* X and Q by two positive integers α and β, respectively. In addition, it allows for variable tolerance in the matching, constrains the matching range by r, and bounds the minimum number of matching elements by δ. The intuition behind allowing gaps in both sequences is to deal with serious humming errors that are likely to occur due to temporary key/tempo loss or significant instant note loss (more than the acceptable tolerance). Thus, we should be able to skip these elements. We will refer to a special case of SMBGT where α and β are set to infinity as SMGT, i.e., no constraints are imposed on the length of the allowed gaps.

3.5.1.1 Computation

Consider an "alignment array" a of size $(|Q|+1)*(|X|+1)$, where Q, X are the compared sequences. $\forall i \in \{1,\ldots,|Q|\}$ and $\forall j \in \{1,\ldots,|X|\}$, the recursive computation for SMGT is:

$$a_{0,j} = 0 \text{ and } a_{i,0} = 0, \tag{3.18}$$

$$a_{i,j} = \begin{cases} a_{i-1,j-1}+1 & \text{, if } \quad q_i \approx_\epsilon^f x_j \\ \max\{a_{i-1,j}, a_{i,j-1}\} & \text{, otherwise.} \end{cases} \tag{3.19}$$

An additional matrix s keeps for each cell $a_{i,j}$ the start point of its best alignment, in $s_{i,j}$, and is updated according to the transitions.

For both SMGT and SMBGT, the computation of a is performed in an online fashion and the space complexity is $O(|Q|)$ as they do not need to store the whole matrix a. Instead, two 1-dimensional arrays are used, *prev*, *cur*, to track the scores (*prev.value* and *cur.value*) and start points (*prev.start* and *cur.start*) of two consecutive columns, $j-1$, j, of a, since having the values of column $j-1$ of a suffices to compute column j. The goal is to be able to match *any* subsequence of Q with *any* subsequence in the database. According to the above recursion, whenever a match occurs the score on the alignment path is increased by 1, otherwise the maximum score of the two adjacent (left, top) cells is inherited with no extra transition penalty. In case of a tie, we choose the transition that corresponds to the subsequence with the most recent start point since this subsequence includes a smaller number of gaps. In SMBGT, the recursion for a is modified to include constraints α and β. Thus, when a mismatch occurs at position (i,j), $a_{i,j}$ stores the largest number of matched elements that can be propagated vertically or horizontally, while not violating α and β. This is checked by an additional step, called *propagation* (Section 3.5.1.2).

Input: query Q, target X, column index j, array cur, and δ.

Output: current best match $best$.

begin

// return the value and start point of the cell with the maximum value in cur.

$\{l_{max}, l_{start}\} = max\{cur\}$;

$l_{len} = j - l_{start} + 1$;

$b_{len} = best_{end} - best_{start} + 1$;

if $best == null \wedge l_{max} \geq \delta$ **then**

$best_{value} = l_{max}$; $best_{start} = l_{start}$; $best_{end} = j$;

end

else if $l_{max} > best_{value} \vee (l_{max} == best_{value} \wedge b_{len} > l_{len})$ **then**

$best_{value} = l_{max}$; $best_{start} = l_{start}$; $best_{end} = j$;

end

end

Algorithm 1: Function $Update()$ for $SMGT$ and $SMBGT$.

The maximum length of the database subsequence with the longest common bounded-gapped subsequence is constrained by r, and the minimum matching score by δ (Section 3.3.2). Notice that during the computation, $best = (best_{value}, best_{start}, best_{end})$ keeps track of the current best solution, with $best_{value}$ being the value of the best match, and $best_{start}$, $best_{end}$ the start and end points of that match, respectively. $best$ is updated as shown in Algorithm 1 taking into account δ. At the end of the computation $best$ corresponds to LCBGS(Q, X). Finally, given K, SMBGT reports the K database sequences, where the K longest common bounded-gapped subsequences occur. This is essentially the solution to the Top-K Subsequence Matching Problem. To keep track of these subsequences, a priority queue S is maintained and updated accordingly using function $Update_{queue}()$. When a new candidate subsequence is found,

Input: query Q, target X, column index j, array cur, functions ϵ_1^f and ϵ_2^f, and match range r.

Output: updated column cur.

begin

  for $i \leftarrow 1$ **to** $|Q|$ **do**

    if $j - cur_i.start + 1 == r$ **then**

      if $q_i \approx_\epsilon^f x_j$ **then** $cur_i.value = 1$;

      ;

      else //determine the appropriate transition and return the value and start point.

      $\{cur_i.value, cur_i.start\} = check(cur, prev)$;

      ;

    end

  end

end

Algorithm 2: Function $Reset()$ for $SMGT$.

S is updated if it contains less than K elements or if the new candidate match has a higher score than any of the K subsequences in S. The main steps of SMBGT are shown in Algorithm 3.

3.5.1.2 Propagation

Two additional arrays, A_{start} and B_{start}, are used to determine the direction of the propagation (left or top). Thus, for each cell (i, j), A_{start} and B_{start} store the latest match positions in X and Q, respectively. Two versions of these arrays (A_{start}^{prev}, A_{start}^{cur} and B_{start}^{prev}, B_{start}^{cur}) are used corresponding to $prev$ and cur, respectively. Suppose that the value for cell (i, j) (i.e., cur_i) is being computed and $propagation()$

is triggered due to a mismatch between q_i and x_j. This function will check whether the value of an adjacent (left or top) cell can be inherited. If $j - A^{prev}_{start}(i) \leq \alpha$ then left propagation is allowed. Similarly, if $i - B^{cur}_{start}(i-1) \leq \beta$, top propagation is allowed. We always choose the propagation that inherits the highest value in matrix a. In case both $prev_i$ (i.e., $a_{i,j-1}$) and cur_{i-1} (i.e., $a_{i-1,j}$) can be propagated, we choose $\max\{prev_i.value, cur_{i-1}.value\}$. If no propagation is possible, $cur_i.value = 0$ and $cur_i.start = 0$, so that another match can start at this point. In case of a tie, we choose the transition that leads to the subsequence with the most recent start point as this subsequence includes a smaller number of gaps.

3.5.1.3 Eliminating Large Matches

Regarding SMGT, due to constraint r it is necessary to perform an additional step (called *Reset*()) in order to avoid expanding matching subsequences whose length is r, and therefore are not going to be included in the final set of top-K matches. After computing *cur* and updating *best*, we scan *cur* to detect those cells that correspond to subsequence matches with length equal to r. This elimination is performed by function *check*(), which returns the new value and corresponding start point for each cell. Hence, for each cell (i, j), if $q_i \approx^f_\epsilon x_j$, then cur_i (that corresponds to that cell) is set to 1. Otherwise, it is checked whether inheriting the value of the left ($prev_{j-1}$) or top (cur_{i-1}) cell may lead to a violation of r. The value of the left cell can be inherited if the subsequence length that corresponds to that cell is less than $r - 1$. The intuition is that if the corresponding length is equal to $r - 1$ this subsequence would have already been reported as a candidate match on *cur* and thus we should not expand it further. Moreover, the value of the top cell can never be equal to r, as the elimination is performed on *cur* from top to bottom. Thus, the value of the top cell can always be inherited. If both values can be inherited, we select the

Input: query Q, target X, gap constraints α and β, functions ϵ_1^f and ϵ_2^f, match range r, and parameter K.

Output: priority queue S.

begin

$S = null$;

for $t \leftarrow 1$ **to** $|DB|$ **do**

$best_{value} = 0$; $best_{start} = 0$;

for $j \leftarrow 1$ **to** $|X_t|$ **do**

for $i \leftarrow 1$ **to** $|Q|$ **do**

if $q_i \approx_\epsilon^f x_j$ **then**

$cur_i.value = prev_{i-1}.value + 1$;

$cur_i.start = prev_{i-1}.start$;

end

else

$cur_i = propagation(i, j, A_{start}, B_{start})$;

end

end

$best = Update(j, cur)$;

$cur = Reset_B(j, cur, A_{start}, B_{start})$;

end

$Update_{queue}(S, best, K)$;

end

end

Algorithm 3: $SMBGT$.

transition with the highest value and in case of a tie the transition that leads to the subsequence with the most recent start point. If no transition is possible, *check*() returns 0 as the cell and start point value. The main steps of function *Reset*() can be

seen in Algorithm 2. In SMBGT the reset function that has to be triggered (called $Reset_B()$) is similar to $Reset()$ with an additional propagation check in case of a mismatch between q_i and x_j. If no propagation is possible the value and start point of the cell are set to 0, otherwise they are updated accordingly. In particular, in the case of a top propagation, the value of the top cell is inherited by the current cell. In the case of a left propagation, it should be ensured that it may not lead to a subsequence that violates r (the length of the subsequence that corresponds to the left cell is less than $r-1$). Finally, if both propagations are possible, we perform the one leading to the subsequence with the highest score and in case of a tie we perform the one that leads to the subsequence with the most recent start point.

3.5.2 Example of SMBGT

Consider the following example: let $Q = (0, -4, 1, 2, -2)$ and $X = (0, 0, -4, 3, 0, 2, -3, 1)$. We want to find the LCBGS(Q, X), with $\alpha = 2$, $\beta = 1$, $\delta = 3$, and $r = |Q| = 5$. For simplicity, we consider only the pitch dimension and do not impose any matching tolerance. We show four matrices: (1) a, which is the alignment array used for the DP computation, (2) s, which contains for each cell of a the start point of the best matching subsequence that ends on that cell, (3) A_{start}, and (4) B_{start}, which are the additional matrices used by function *propagation*(). Following Equation 3.18 and Algorithm 3, the first 6 columns of all four matrices are shown in Figure 3.2 (a). At this phase, column 6 contains cells that will trigger function $Reset_B()$. Consider row 5 of column 6. The start point of the subsequence that corresponds to that row is at position 2, which gives a subsequence of length $6-2+1 = 5 = r$. Thus, this cell should be reset. Since $q_i = x_j = 2$ (match), the new value of that cell should be 1. Let us check the next row of column 6. The length of the corresponding subsequence is now $6-2+1 = 5 = r$, however in this case $q_i \neq x_j$, thus we should check whether

any propagation is possible. Regarding the left propagation, $j - A^{prev}_{start}(i) = 6 - 0 > \alpha$; hence left propagation is not allowed. Also, $i - B^{cur}_{start}(i-1) = 5 - 4 = 1 = \beta$; hence top propagation is allowed, and the new value of cell $(5, 6)$ is set to 1. Notice that s, A_{start}, and B_{start} are updated accordingly. The new matrices are now reset and are shown in Figure 3.2 (b). Clearly, $best_{value} = 3$, $best_{start} = 2$, and $best_{end} = 6$.

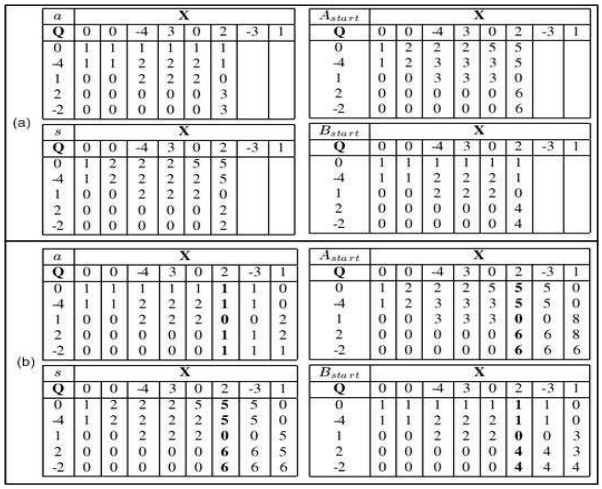


Figure 3.2: DP matrices for SMBGT (a) before and (b) after reset.

The appropriate tuning of the parameters used in SMBGT depends on the application domain. Learning approaches such as leave-one-out cross validation on a sample of training queries could be used for parameter tuning.

According to the algorithm described in Section 3.5.1, the computational time complexity of SMBGT is $O(|Q||X|)$, which may be prohibitive for large databases. Moreover, due to the fact that this method is not metric, the task of speeding up similarity search under SMBGT is even tougher. Consequently, there is a need for an efficient indexing approach, which is proposed in the next section.

3.6 ISMBGT: Indexed Subsequence Matching with Bounded Gaps and Tolerances

Next, we present ISMBGT, a novel approach for speeding up similarity search under SMBGT. ISMBGT exploits an embedding-based technique for indexing the database sequences, and, given a query sequence, works in a filter-and-refine manner.

3.6.1 SMBGT Embeddings

The task at hand is to identify the top-K subsequence matches to a query Q in a database of sequences DB. The naive solution to this problem is to employ brute-force search by using the SMBGT method described in the previous section.

Here, we present a more efficient method, which is based on defining a novel embedding function H, called *SMBGT embedding*. The key novelty of this embedding function is that it is specifically tailored for subsequence matching under SMBGT. Every element x_j of each database sequence X is mapped into an n-dimensional vector, called *SMBGT database sequence embedding*, and each query Q is also mapped into an n-dimensional vector, called *SMBGT query embedding*. This mapping is performed via the use of a set of relatively short sequences, which we call *reference sequences*.

Let R be a reference sequence, Y be a target sequence (either database sequence or query), and $Y[i:j]$ be a subsequence of Y. We define $S_{|R|,j}(R, Y[i:j])$ to be the highest value of the last column of the alignment table α when we compute (in a dynamic programming manner) $LCBGS(R, Y[i:j])$ using SMBGT:

$$S_{|R|,j}(R, Y[i:j]) = max_{t=0,\ldots,|R|}\{\alpha_{t,|Y[i:j]|+1}\} \ . \tag{3.20}$$

We shall use R to define a 1-D SMBGT embedding.

Definition 3. (*1-D SMBGT embedding*) Given a reference sequence R and a target sequence Y, function H^R is a 1-D SMBGT embedding that maps any subsequence $Y[i:j]$ of Y into a real number using R as follows:

$$H^R(Y,i,j) = S_{|R|,j}(R, Y[i:j]) \ . \tag{3.21}$$

Using the above definition, we can compute a 1-D SMBGT database sequence embedding as follows:

$$H^R(X,i,j) = S_{|R|,j}(R, X[i:j]) \ . \tag{3.22}$$

Regarding i, it is set to $j - c + 1$, where $c > 0$ is a constant determined by $|Q|$. Note that, in order to define i, it has to hold that $j \geq c$. However, if $|X| < c$ then we set $i = 1$ and $j = |X|$. The instantiation for c is given in Section 3.7.4.

Similarly, a 1-D SMBGT query embedding is computed as follows:

$$H^R(Q,i,|Q|) = S_{|R|,|Q|}(R, Q[i:|Q|]) \ . \tag{3.23}$$

In this case we set $i = |Q| - c + 1$.

Consequently, for each reference sequence R, each database sequence is mapped to $|X| - c + 1$ values, if $|X| \geq c$, or one value if $|X| < c$. Note that for the case of the query, by definition, it holds that $|Q| \geq c$.

Naturally, instead of picking a single reference sequence R, we can pick multiple reference sequences to create a multi-dimensional embedding, as defined below.

Definition 4. (*n-D SMBGT embedding*) Given a set of n reference sequences $\mathcal{R} = \{R_1, \ldots, R_n\}$ and a target sequence Y, function $H^{\mathcal{R}}$ is an n-D SMBGT embedding that maps any subsequence $Y[i:j]$ of Y using $\mathcal{R}$ to the following vector:

$$H^{\mathcal{R}}(Y,i,j) = \{S_{|R_1|,j}(R_1, Y[i:j]), \ldots, S_{|R_n|,j}(R_n, Y[i:j])\} \ . \tag{3.24}$$

Hence, using Equations 3.22 and 3.23 we can compute the n-D SMBGT database sequence and query embedding vectors, respectively, as follows:

$$H^{\mathcal{R}}(X,i,j) = \{H^{R_1}(X,i,j),\ldots,H^{R_n}(X,i,j)) \ . \tag{3.25}$$

$$H^{\mathcal{R}}(Q,i,|Q|) = \{H^{R_1}(Q,i,|Q|),\ldots,H^{R_n}(Q,i,|Q|)\} \ . \tag{3.26}$$

3.6.2 Properties of SMBGT Embeddings

The construction of the SMBGT embeddings is performed in such a way that if Q is similar to a subsequence of X starting at x_i and ending at x_j, and if R is some reference sequence, then $H^R(Q,i,|Q|)$ is likely to be similar to $H^R(X,i,j)$. Using the same argumentation as in Papapetrou et al. [98], we can see that:

- if Q appears exactly as a subsequence $X[i:j]$ in X, it holds that $H^R(Q,i,|Q|) = H^R(X,i,j)$, given that the optimal alignment path computed by SMBGT when matching R with $X[i:j]$ does not start before position i, which is the position in X where the appearance of Q starts, and
- if $X[i:j]$ is a slightly perturbed version of Q in X, then $H^R(Q,i,|Q|) \approx H^R(X,i,j)$. In other words, the perturbed version of Q has been obtained, for example, by adding some noise to each query value q_t, $t = 1,\ldots,|Q|$, of Q.

Thus, little tweaks in the values of Q should slightly affect the difference between the values of $H^R(Q,i,|Q|)$ and $H^R(X,i,j)$. This claim becomes even stronger when more reference sequences are used (see Papapetrou et al. [98]), which is the case in our approach. Unfortunately, due to the non-metric nature of SMBGT, there exists no approximation method that can make any strong theoretical guarantees in the presence of perturbations along the temporal axis. In order for the proposed embeddings to provide good retrieval accuracy, the following *statistical* property has to hold empirically:

Property: (*Embedding Similarity*) Let j_{opt} be the position in X where the optimal SMBGT subsequence match of Q in X ends. Then, given some random position $j \neq j_{opt}$, it should be very likely that $H^{\mathcal{R}}(Q, i, |Q|)$ is closer to $H^{\mathcal{R}}(X, i, j_{opt})$ than to any other $H^{\mathcal{R}}(X, i, j)$.

The above statistical property can be established by embedding optimization. It can be seen that the quality of the SMBGT embeddings depends on the selection of the reference sequence set $\mathcal{R}$. In Section 3.6.6, we describe a way of doing this, which is also one of the key differences of our framework against existing state-of-the-art embedding-based filter-and-refine techniques.

3.6.3 A filter-and-refine Framework

Applying brute-force SMBGT on the database for a given query would be prohibitively computationally expensive for large databases. Our goal is to improve retrieval efficiency with very small loss in retrieval accuracy. Towards this objective, the design of ISMBGT enables a filter-and-refine nearest neighbor retrieval [92] using the SMBGT embeddings defined in the previous section. ISMBGT employs a filter step, where a small set of candidate database subsequences are selected, which are then passed to the refine step to perform the costly SMBGT computation. The substantial speedup is achieved by keeping the number of filtered sequences quite small while allowing for very small loss in accuracy.

A pre-processing step is needed to create the SMBGT database embeddings. In particular, ISMBGT first performs a one-time offline step computation, where vector $H^{\mathcal{R}}(X, i, j)$ is computed for every allowable position j of each database sequence X. Computing the set of all n-D SMBGT database embeddings $H^{\mathcal{R}}(X, i, j)$, for $j = c, \ldots, |X|$ (with $|X| \geq c$) takes time $O(|X| \sum_{l=1}^{n} |R_l|)$. This process is repeated for all database sequences and the resulting set of embeddings corresponds to the

embedding of the whole database. This set is stored and ISMBGT is then ready for receiving queries.

At query time, qiven a query sequence Q, ISMBGT performs three steps:

- *Query embedding step*: The SMBGT query embedding $H^{\mathcal{R}}(Q, i, |Q|)$ is computed online, by applying the SMBGT method n times, one time for each of the n reference sequences. The total running time is $O(|Q| \sum_{l=1}^{n} |R_l|)$. This time is typically negligible compared to running SMBGT between Q and all sequences X in DB, which takes $O(|Q| \sum_{l=1}^{N} |X_l|)$ time, because we enforce $\sum_{l=1}^{n} |R_l| << \sum_{l=1}^{N} |X_l|$.
- *Filter step*: $H^{\mathcal{R}}(Q, i, |Q|)$ is compared to $H^{\mathcal{R}}(X, i, j)$ of each database sequence, for $j = c, \ldots, |X|$, and some j's are chosen such that $H^{\mathcal{R}}(Q, i, |Q|)$ is very similar to $H^{\mathcal{R}}(X, i, j)$. Specifically, some pairs of database sequences X and positions j in those sequences (we denote them as (X, j)) are selected according to the distance between each $H^{\mathcal{R}}(X, i, j)$ and $H^{\mathcal{R}}(Q, i, |Q|)$. These are the *candidate endpoints* of the subsequences that best match with Q.
- *Refine step*: For each such j, and for a matching range r, SMBGT is run between Q and $X[j - r + 1 : j]$ to find the best subsequence match. At the end of this step the subsequences with the K highest SMBGT scores, or else the K best matching subsequences to Q, will have been identified, thus solving the Top-K Subsequence Matching Problem.

It is obvious that if we are able to choose a small number of such promising subsequences $X[j - r + 1 : j]$, evaluating only those areas will be much faster than running SMBGT between Q and the whole database.

3.6.4 Filter Step

3.6.4.1 Original Scheme

We first select the best k reference sequences, i.e., the reference sequences that provide the k highest similarity scores in $H^{\mathcal{R}}(Q, i, |Q|)$. Then, for every (X, j) we compute the Euclidean distance between $H^{\mathcal{R}}(Q, i, |Q|)$ and $H^{\mathcal{R}}(X, i, j)$ using only the dimensions corresponding to the selected reference sequences. Finally, all database positions (X, j) are ranked in increasing order of the distance between $H^{\mathcal{R}}(X, i, j)$ and $H^{\mathcal{R}}(Q, i, |Q|)$, so that they can be further used at the refine step.

The time complexity of this step is $O(k \sum_{l=1}^{N} |X_l|)$, which suggests that for large database sizes the filter step can still be expensive. Hence, we next present a sampling technique which, as we confirm in our experiments, can achieve significant speedups.

3.6.4.2 Speeding Up the Filter Step

To reduce the computational cost of the filter step, ISMBGT performs sampling over the vector space of each SMBGT database sequence embedding. The intuition is that embeddings are constructed in a way that embeddings of nearby positions, such as $H^{\mathcal{R}}(X, i, j)$ and $H^{\mathcal{R}}(X, i, j+1)$, tend to be very similar.

We choose a sampling parameter s, and then sample uniformly one out of every s vectors $H^{\mathcal{R}}(X, i, j)$ in each X. In other words, we only store vectors:

$$H^{\mathcal{R}}(X, i, 1), H^{\mathcal{R}}(X, i, 1+s), H^{\mathcal{R}}(X, i, 1+2s), \ldots.$$

Given $H^{\mathcal{R}}(Q, i, |Q|)$, we compute its distance with the vectors that we have sampled, using only the k dimensions that correspond to the reference sequences that provide the best k similarity scores in $H^{\mathcal{R}}(Q, i, |Q|)$ (similar to the original scheme). If for a database position (X, j) its vector $H^{\mathcal{R}}(X, i, j)$ was not sampled, then we assign to

that position the distance between $H^{\mathcal{R}}(Q, i, |Q|)$ and the vector that was actually sampled among $\{H^{\mathcal{R}}(X, i, j - \lfloor s/2 \rfloor), \ldots, H^{\mathcal{R}}(X, i, j + \lfloor s/2 \rfloor)\}$.

The time complexity of this step is $O(k \sum_{l=1}^{N} \lceil \frac{|X_l|}{s} \rceil)$. In our experiments, we observed that sampling achieves significant runtime improvements compared to brute-force search, without essential loss in accuracy.

3.6.5 Refine Step

As mentioned above, the filter step ranks all database positions (X, j) in increasing order of the distance, or estimated distance when we use approximations such as sampling, between $H^{\mathcal{R}}(X, i, j)$ and $H^{\mathcal{R}}(Q, i, |Q|)$. The refine step then evaluates a percentage *perc* of the total number of candidate positions (which are ranked), where *perc* is a system parameter that provides a trade-off between retrieval accuracy and efficiency.

Since candidate positions (X, j) actually represent candidate *endpoints* of a subsequence match, we can evaluate each such candidate endpoint by performing the SMBGT method from a startpoint determined by r up to this endpoint. Specifically, if j_{end} is the endpoint of a potential match, we run the SMBGT method between Q and $X[j_{\text{end}} - r + 1 : j_{\text{end}}]$. In other words, SMBGT is used to find the best matching score and the corresponding subsequence for that candidate endpoint. We have to mention that if we do not put any constraint on the value of the matching range parameter r, or else set it to infinity, SMBGT will be evaluated from the beginning of the database sequence X. However, subsequences of X that are much longer than Q are very unlikely to be good matches for Q, since there will be many unmatched elements creating large gaps when aligning the two sequences with SMBGT.

After all similarity scores between Q and candidate positions (X, j) have been computed, we sort them in decreasing order and check if the correct/targeted sequence

is included in the returned results. If this is the case, then the rank of the query is the position of the correct sequence. Otherwise, the rank of the query is the rank of the targeted sequence when we apply the ISMBGT to all the endpoints, and not just the *perc* of them.

The time complexity of the refine step is $O(r\ |Q|\ perc\ \sum_{l=1}^{N} |X_l|)$.

3.6.6 Query-optimized reference sequences

It can be easily seen that the construction of the SMBGT query embedding is highly dependent on the length difference between the query and the reference sequences. If the length difference is high (the reference sequence is either much smaller or much longer than the query), then this will produce a low SMBGT similarity score. In order to guarantee that the Embedding Similarity Property holds, we should always choose reference sequences that are shorter than the queries but not too much shorter either. This has also been argued in Papapetrou et al. [98].

Hence, we employ a novel technique for building the SMBGT query embeddings. We first introduce a lower and an upper bound of the possible query lengths that our filter-and-refine framework may accept, denoted as L_Q and U_Q, respectively. Next, we split the query lengths into d' intervals, which results in a set of d' *buckets* $\mathcal{B} = \{b_1, \ldots, b_{d'}\}$. Each backet b_i corresponds to interval $[uv^{i-1}, uv^i)$, for $i = 1, \ldots, d'$, where $u = L_Q$ and $uv^{d'} = U_Q$. The resulting set of buckets is the following:

$$\mathcal{B} = \{[u, uv), [uv, uv^2), [uv^2, uv^3), \ldots [uv^{d'-1}, uv^{d'})\} \tag{3.27}$$

Then, we assign a set of reference sequences to each bucket b_i that satisfies the corresponding length requirement of b_i. Specifically, for each bucket b_i, the number of reference sequences is set to q, and their length is defined as a percentage p of the

lower bound of the bucket, i.e., puv^{i-1}. This process is performed offline as well as the construction of the embedding of the whole database.

At query time, given Q, we identify the bucket $b_i = [uv^{i-1}, uv^i)$, such that

$$uv^{i-1} \leq |Q| < uv^i.$$

The SMBGT query embedding is then created using the set of q reference sequences that correspond to bucket b_i.

3.7 Experiments

In the first part of our experiments (Sections 3.7.2 and 3.7.3) we show the superiority of SMGT and SMBGT in terms of accuracy over several DP-based methods and the HMM-based method on QBH, using both synthetic and hummed queries. In the second part (Section 3.7.4) we show the usefulness of the proposed indexing approach, ISMBGT, when compared to the brute-force SMBGT in terms of retrieval accuracy and runtime. Finally, in Section 3.7.5 we highlight the main observations and conclusions from our experimental evaluation.

3.7.1 Experimental Setup

3.7.1.1 Database

We created a music database of 5,643 freely available on the web MIDI files that cover various music genres, such as Blues, Rock, Rock 'n' Roll, Pop, Classical, Jazz, and also themes from movies and tv series. For each MIDI (comprising 16 channels) and channel, we extracted the highest pitch at every time click (*all-channels extraction* [20])[2]. Then, we converted tuples ⟨pitch, click⟩ to ⟨pitch interval, IOIR⟩, resulting

[2]In the extraction process we excluded channel 10, since it is used for drums and cannot offer any musical information in QBH.

in 40,891 sequences (that is the $|DB|$) and a total of 13,455,603 tuples. This pre-processing procedure was done offline and only once, guaranteeing that there is no chance of missing a melody existing in any channel of a song.

3.7.1.2 Synthetic Queries

Six synthetic query sets (100 queries per set) of lengths between 13 and 137 were generated: Q_0, $Q_{.10}$, $Q_{.20}$, $Q_{.30}$, $Q_{.40}$, and $Q_{.50}$. Q_0 contained 100 exact segments of the database, while $Q_{.10} - Q_{.50}$ were generated by adding noise to each query in Q_0. For all queries of Q_0 we randomly modified 10, 20, 30, 40, and 50% of their corresponding time series in both dimensions. The pitch interval of the modified elements was changed by $\pm z \in [3, 8]$ (integer), as we wanted the noise to range within one octave. This simulates the error performed by a human when singing a song by memory as well as the intrinsic noise that may be added by any audio processing tool. An erroneous interval of 3 to 8 semitones, i.e., 1.5 to 4 tones, is very reasonable for QBH. Regarding IOIR, each q_i^2 was modified by $\pm z \in [2, 4]$ (real), so that several reasonable variations of duration ratios could be simulated, and also be outside the bounds described by Equation 3.6, avoiding any bias in favor of SMBGT and SMGT. In case of a negative value in IOIR, it is reset to a very small real positive value, as duration ratios should be positive. Moreover, in all noisy query sets we allowed at most 3 consecutive elements to be replaced with noisy values. This is because in QBH we do not expect to have too many consecutive matching errors. We note that noise was added to existing query elements without any insertions or deletions.

3.7.1.3 Hummed Queries

To evaluate the methods in QBH in a real application scenario we also experimented with a set of 100 hummed queries of lengths between 14 and 76. In our setting,

4 males were asked to hum 25 songs each. Two of them were musically trained with middle and low level studies on the piano and the guitar, while the third and fourth had no musical background. The users were asked to sing close to a microphone and avoid singing with lyrics. The hummed melodies were then converted to MIDI using the *Akoff music composer-version 2.0* [3], a well-known tool commonly used for evaluating QBH systems (e.g., by Zhu et al. [61]). All-channels extraction was applied to the queries to obtain the same representation with the database. The query set covered several genres of music, such as Classical ("Für Elise"), Blues ("Hideaway"), Jazz ("Strangers in the Night"), Rock 'n' Roll ("Rock Around the Clock"), Rock ("Fly Away"), Country ("Hey Good Lookin"), and romantic songs ("What a Wonderful World").

An acute reader may notice that the number of people who voluntarily helped in the construction of the hummed query set is not quite large. However, we have to mention that creating such a set is not a trivial task, since selecting the final set of hummed queries involved much manual process. Apart from the mistakes that a user can make, any recording procedure may introduce noise in both pitch and duration. After listening to the MIDI of each hummed song, noise had been introduced, especially in pitch. Consequently, users had to hum each song several times before selecting the version with the least amount of noise, i.e., the one whose melody sounded as close to the target as possible according to them.

3.7.1.4 Evaluation

In the first part of our experimental evaluation (Sections 3.7.2 and 3.7.3) we show the comparison among SMBGT, SMGT, the five DP-based methods and the probabilistic-based approach that were described in Section 3.4, i.e., Edit distance-

[3] http://www.akoff.com/music-composer.html.

based, SPRING, DTW_s, DTW_c, Il. et al., and HMM-based. Edit distance was slightly modified to deal with LogIOIR and quantizations (Section 3.4.1), while for Il. et al. a more elastic version was used that is suitable for subsequence matching and supports both constant and variable error tolerance. In addition, the whole query matching requirement is eliminated, and the rationale behind this is that it would be a too tight constraint to demand all hummed query elements to match in X. Moreover, both Il. et al. and SPRING were modified to allow for r [79]. First, we present the robustness of all methods with respect to noise using the synthetic query sets (Section 3.7.2). Secondly, the methods that achieved a reasonably high recall ($> 90\%$) even for high noise levels (50%) were further tested on hummed queries where the noise level can be much higher (Section 3.7.3).

The performance is evaluated in terms of *recall*, *mean reciprocal rank (MRR)* [100], and *runtime*. Recall is the percentage of queries for which the correct answer is among the top-K returned results. MRR is the mean inverse rank of all queries (of a query set) in their top-K results. Thus, its value belongs to $[0, 1]$. If the right answer is not included in the results, then the inverse rank is 0. The rank of a query is the number of matches (i.e., database sequences) with similarity/distance value at least as high/low as that of the correct match (including the correct match). Both measures are essential for the evaluation of a QBH method, as recall shows how successful the method is in finding the correct answer among the top-K, whereas MRR indicates if there is room for improvement in terms of recall when decreasing K. For all methods all variations and parameter settings were tested [79], and here we report those variations that achieved the best performance.

In the second part of our experiments (Section 3.7.4) we evaluated the performance of ISMBGT compared to that of brute-force SMBGT for the five noisy synthetic query sets and the hummed query set in terms of recall, runtime, and *ef-*

ficiency, which is a particularly useful measure that influences the retrieval runtime. Efficiency is defined as the ratio of the number of database elements that are evaluated during the refine step by SMBGT to the length of the database.

Experiments were run on an AMD Opteron 8220 SE processor at 2.8GHz, and implemented in C++.

3.7.2 Evaluation of Brute-Force SMBGT on Synthetic Queries

3.7.2.1 Parameters, Tolerance, and Representation

For the methods that consider r in their computation (SPRING, Il et al., SMGT, and SMBGT) we set $r = |Q|$, as due to the construction of the query sets, the desirable match does not exceed that value. Taking into account the noise levels of the query sets, in SMBGT and SMGT δ was set to 0.9, 0.7, 0.6, 0.5, 0.35, and 0.3 times the length of each query in the six query sets, $Q_0 - Q_{.50}$, respectively. We selected these gradually decreasing δ values, so as to be elastic enough as noise increases, and avoid false dismissals. We experimented with $\alpha = \beta = 3$, as we know that the maximum number of gaps in all query sets is 3. We tested several values for K, from 5 to 150, for all synthetic query sets. In QBH, however, high values of K may not be practical as users may not be willing to look at a large number of candidate songs to identify the targeted one; we chose a reasonable value, $K = 20$, to report the accuracy regardless of noise level.

For pitch, we tested constant and variable absolute and relative tolerances. For duration, we experimented with variable error tolerance (Equations 3.6 - IOIR and 3.7 - LogIOIR). For all synthetic query sets and pitch SMBGT achieved its best accuracy when using variable absolute tolerance (Equations 3.1 and 3.4) with $t = 0.2$, SMGT when using variable absolute tolerance with $t = 0.5$, and Il. et al. when using constant

relative tolerance (Equation 3.2) with $\epsilon_1^f = 1$. All the above accuracies where obtained using IOIR for duration. The best accuracy for the HMM-based method was achieved for $M = 5$ states, and the observation distribution of the states we experimented with was Gaussian (which is common) [79]. The results of the performance evaluation of SMBGT and SMGT with respect to accuracy are shown in the upper part of Figure 3.3.

Regarding the representation of music pieces, as the noise level increases the representations achieving the highest recall per DP-based method are a subset of the representations of lower noise levels. Moreover, representations ⟨mod12, IOIR⟩ and ⟨pitch interval, IOIR⟩ appear most in all synthetic query sets. For the HMM method, the representation used is ⟨mod12, LogIOIR in $[-2, 2]$⟩. These observations show that the simpler the representation the more promising it seems to be in QBH [79].

3.7.2.2 Accuracy

Regarding the 100 exact queries (Q_0), all DP-based methods achieved 100% recall with MRR 1 for top-5, except for DTW_s, which did not exceed a recall of 96% even for $K = 250$. The HMM-based method, achieved 96% recall for top-5 with MRR 0.95, and recall 100% for top-150. SMBGT, SMGT, and Edit distance performed best, and behaved similarly for all query sets. Even for $Q_{.50}$ their recall is 97, 96, and 97%, respectively. The reason for the high recall of Edit distance is that if two elements do not match, it will increase the total matching score by at most 1, while for the remaining intact elements this score will not be affected. On the contrary, the recall of all other competitor methods degrades with noise. SPRING and the HMM-based method behave similarly for $Q_{.10}$, $Q_{.20}$, and $Q_{.30}$ presenting a recall of more than 91%, but further increasing the noise level results in a smooth degradation

for SPRING and a sharp one for the HMM-based method. For $Q_{.50}$ their recall is only 75 and 56%, respectively. The accuracy of DTW$_c$ and Il. et al. degrades very sharply when adding noise, achieving for $Q_{.40}$ 27 and 53% recall, and for $Q_{.50}$ 7 and 32%, respectively. The latter method presents such behaviour, as it will sooner stop its computation when not being able to find a match for a query element. SMBGT significantly outperforms Il. et al. with $\alpha = \beta$ due to its additional ability to skip query elements. DTW$_s$ performs worst for all noise levels (0% recall for $Q_{.50}$), and this behaviour, along with that of DTW$_c$, is justified by the fact that they implicitly embed time by allowing it to adjust locally, and they are unable to skip non-matching elements, as they force them to align. For both of them, the value of parameter c (Equation 3.15) achieving their best recall was 2. SPRING explicitly accounts for duration information in its computation, thus it can tolerate higher noise levels as opposed to DTW$_s$ and DTW$_c$. Referring to MRR, the same conclusions hold, with SMBGT, SMGT, and Edit distance remaining close to 1 for all noise levels, which is expected as the other competitors identify fewer correct answers in the top-K results. An overview of these findings is shown in Figure 3.3 (upper-part).

Table 3.1: Recall of SMGT for various top-K and ranges for the hummed queries.

	Recall (%)					
r	$\|Q\|$	$1.1\|Q\|$	$1.2\|Q\|$	$1.5\|Q\|$	$2\|Q\|$	∞
$K = 5$	32	31	34	31	30	0
$K = 10$	38	40	42	36	40	0
$K = 20$	41	46	48	44	42	0
$K = 50$	47	52	62	57	56	0

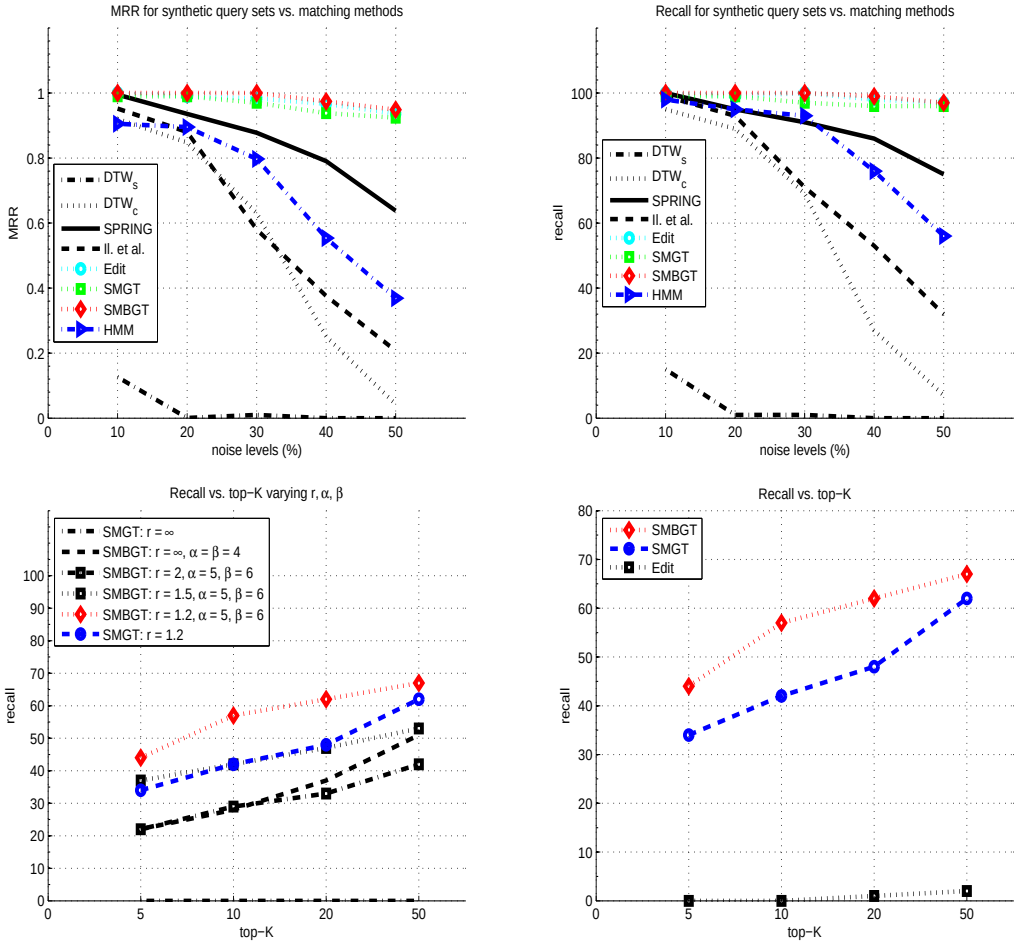


Figure 3.3: (up-left) Accuracy of all DP and HMM-based methods for $K = 20$ in terms of MRR for synthetic queries; Recall of SMBGT and SMGT for: (up-right) synthetic queries vs. DP and HMM-based methods for $K = 20$; (bottom-left) hummed queries varying $\alpha, \beta, r,$ and K; (bottom-right) hummed queries vs. Edit distance varying K, when $r = 1.2$ for both SMBGT and SMGT, and $\alpha = 5$, $\beta = 6$ (SMBGT).

Table 3.2: Recall and MRR of the proposed methods vs. Edit distance for the hummed queries ($K = 50$).

	Accuracy		
Methods	Recall (%)	MRR	Tol.
SMBGT	67	0.3661	abs. $t = 0.2$
SMGT	62	0.2956	abs. $t = 0.25$
Edit	2	0.0012	-

3.7.2.3 Runtime

In Figure 3.4 we show the average execution time for all methods per query length conforming with the complexities of the methods.

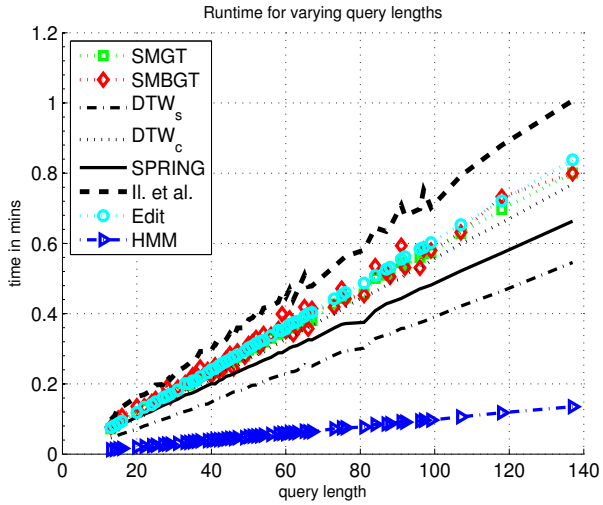


Figure 3.4: Average execution time for all methods per query length.

3.7.3 Evaluation of Brute-Force SMBGT on Hummed Queries

The methods that showed to be noise-tolerant even for high noise levels of 50%, i.e., SMBGT, SMGT, and Edit distance, were further evaluated for the hummed query set, since possibly none of the elements of the correct targeted song will be intact, due to humming errors and noise introduced by the recording procedure.

3.7.3.1 Parameters, Tolerance, and Representation

We experimented with parameters r, α, and β, for $\delta = 0$. First, the effect of r in SMGT, where no constraint is imposed on the number of consecutive gaps, was studied. Increasing r, starting from $r = |Q|$, we observe that the recall of SMGT

increases, until $r = 1.2 * |Q|$, after which the recall degrades. Interestingly, setting $r = \infty$ leads to a recall of 0%. This is not surprising, though, since increasing r without any additional constraint in the number of gaps results in a larger number of erroneous candidate matches. We also studied the influence of α and β for the extreme case of $r = \infty$ for SMBGT. Several pairs of α, β were tested, and the recall was significantly improved [79]. The best recall value was achieved for $\alpha = \beta = 4$ (51% for $K = 50$), as shown in Figure 3.3 (bottom-left). Then, in order to capture the impact of r combined with α and β, we tested the same pairs of values and we gradually decreased the value of r. The best accuracy was achieved for $\alpha = 5$, $\beta = 6$, and $r = 1.2 * |Q|$, verifying the need for skipping elements in both target and query sequences. In Figure 3.3 (bottom-left) we also show how accuracy is improved when varying r from 1.2 to 2, for $\alpha = 5$ and $\beta = 6$.

In Figure 3.3 (bottom) we can see the results for SMBGT and SMGT, using variable absolute tolerance (Equations 3.1 and 3.4), with $t = 0.2$ and $t = 0.25$, for pitch, respectively, and variable error tolerance (Equation 3.6 - IOIR) for duration. For any tolerance type, small constant tolerance and small values of t for variable tolerance made our methods perform better than for greater values, with variable tolerance being better. Moreover, variable absolute tolerance always outperformed variable relative tolerance in all experiments. For example, for $t = 0.2$ and $K = 10$, relative tolerance was 33% worse than absolute. Regarding Edit distance, no matching tolerance can be defined.

The representation used for the music pieces and the hummed queries that led to highest accuracies was ⟨pitch interval, IOIR⟩.

3.7.3.2 Accuracy

Figure 3.3 (bottom-right) shows that SMBGT and SMGT are at least 30 times higher in recall than Edit distance for $K = 50$, while SMBGT achieves 10% higher recall than SMGT for $K = 5$, and 15% for $K = 10$ and $K = 20$. The recall of Edit distance is 0% even for $K = 10$. These values are achieved for $r = 1.2$, $\alpha = 5$ and $\beta = 6$ (for SMBGT), and $\delta = 0.1 * |Q|$ for all K. Regarding MRR, SMBGT outperforms Edit distance by more than two orders of magnitude for $K = 50$, while SMGT achieves worse MRR than SMBGT, as these measures are influenced by the recall. Also, the values of MRR for $K < 50$ are very close to those of $K = 50$. Increasing $K > 50$ may improve the recall of all methods, though, trading K for higher recall will increase retrieval cost as more database sequences will be reported. Our goal, in QBH, is to achieve high recall by reporting as few candidates as possible. Hence, for $K =$ 5-50 SMBGT and SMGT clearly offer higher recall.

Finally, we tested the impact of δ on recall for the best combination of parameters for SMBGT. Increasing δ, even for $\delta = 0.5 * |Q|$ and $K = 50$ the recall does not decrease, while further increasing it makes recall worse. For example, for $\delta = 0.6 * |Q|$ it degrades to 28%, and for $\delta = 0.7 * |Q|$ to 9%, when $K = 5$. This behavior indicates that the recall on the hummed queries for which the correct song appeared in the top-K results was not influenced by requesting more elements of the targeted sequences to match to theirs, until $\delta = 0.5 * |Q|$. In other words, these hummed queries were very similar to the targeted songs, leading us to the conclusion that if, in QBH, the users are singing well (both in pitch and time), δ can be set, for example, to $0.5 * |Q|$, resulting in fewer false positive candidates. We note here though that, in order to maximize the accuracy, δ can be set to a very small value, e.g., $0.1 * |Q|$, since the

Table 3.3: Buckets statistics.

Interval Notation	**Interval**	c	$\lvert R \rvert$	**Hummed Queries**	**Synthetic Queries**
b_1	$[13, 21)$	13	10	15	25
b_2	$[21, 33)$	21	17	30	90
b_3	$[33, 53)$	33	26	39	180
b_4	$[53, 86)$	53	42	16	135
b_5	$[86, 138)$	86	69	-	70

target group of users may include people with not only good but also bad singing capabilities.

3.7.4 Evaluation of ISMBGT on Synthetic and Hummed Queries

3.7.4.1 Reference Sequences

We followed the process described in Section 3.6.6 to select reference sequences for the construction of the SMBGT embeddings. We first created the set of query length buckets $\mathcal{B}$. Since query lengths are in $[13, 137]$, we set $L_Q = 13$ and $U_Q = 138$. In addition, we set $d' = 5$, which means that we considered 5 query length intervals. Thus, $v \approx 1.60$. Next, for each bucket we set $q = 1,000$ and $p = 0.8$. Thus, we randomly selected $1,000$ sequences from DB with their lengths being 80% of the lower bound of the bucket. As a result, $n = 1,000$, i.e., we constructed 1,000-dimensional SMBGT embeddings. We should note that the construction of the SMBGT database sequences embeddings is done offline. The value of c needed when creating the SMBGT embeddings is set to the lower bound of the bucket that the length of a query belongs to. Hence, for our experiments c was 13, 21, 33, 53, and 86 for the 5 buckets, respectively. We will use the terms (query length) intervals and buckets interchangeably. The lower and upper bounds of the buckets, the lengths of the reference sequences, and the c values are shown in Table 3.3.

3.7.4.2 Representation, Parameters, and Implementation details

For the experimental evaluation of ISMBGT and its comparison with the brute-force SMBGT (which for the remainder of this section we will also denote as BF), the representation used is ⟨pitch interval, IOIR⟩. This is because, as shown in Section 3.7.2.1 and 3.7.3.1, this representation leads to highest accuracies for both synthetic and hummed query sets compared to all other possible representations.

In order to be fair when comparing brute-force SMBGT with ISMBGT, we set the parameters of SMBGT as follows. For the synthetic queries the number of gaps that could be skipped in the target sequences and the query was 3 ($\alpha = \beta = 3$) and the subsequence matching range r was set to $|Q|$. No matching tolerance was imposed, so as to enforce the intact elements of the queries to be matched with the corresponding ones from the targeted subsequences, while the remaining elements were skipped through the α and β parameters. The minimum number of elements that have to match between the query and the target sequence δ was set to $0.1 * |Q|$. For the hummed queries, variable absolute tolerance with $t = 0.2$ was applied for pitch (Equations 3.1 and 3.4) and variable error tolerance (Equation 3.6 - IOIR) for duration, $\alpha = 5$, $\beta = 6$, $r = 1.2 * |Q|$, and $\delta = 0.1 * |Q|$ so as to have at least some elements of Q to match with the subsequences returned in the top-K results.

With regard to the percentage of endpoints evaluated at the refine step of ISMBGT, after experimenting with $perc = 0.1-0.9\%$ with step 0.2 and $perc = 1-5\%$ with step 2, we set it to 1%. This choice accounts for the trade-off between accuracy and runtime. Lower values of *perc* did not provide satisfying recall, although speedup improvements were higher than for $perc = 1\%$, while greater values significantly decreased speedup. $perc = 1\%$ makes ISMBGT at least 2 times faster than BF

(Section 3.7.4.4), which is very significant in a real QBH scenario, especially if we consider that it leads to very high recall values.

At the refine step we keep for each sequence X corresponding to a channel an array of size equal to $|X|$, which is initialized with zeros. Then, for each candidate endpoint j_{end} of the database, i.e., that belongs to the *perc* of the sorted endpoints, we put the value one to all the positions of the array starting from j_{end} and going backwards up to position $j_{\text{end}}-r+1$. After that operation, we trace each sequence that has non zero values in its array and perform SMBGT between Q and the segments that were filled in with ones. If there are more than one segments for X, then we keep the best similarity score that was obtained after applying SMBGT to all of them as the similarity score for X. The similarity scores of the channel sequences are finally sorted and it is checked whether any channel of the correct/targeted song is included in these channels. If there exists a channel of the correct song, then the rank of the query is the rank of the channel of the correct song with the highest similarity score (worst in case of similarity score ties). Otherwise, the rank of the query is the (worst in case of ties) rank of the closest channel of the song (the one with the highest similarity score) when we apply the ISMBGT to all the endpoints, and not just the *perc* of them.

3.7.4.3 Accuracy

We first evaluated BF and ISMBGT in terms of accuracy on the five noisy synthetic query sets and the hummed query set (given in Section 3.7.1). For completeness, we mention that the approach followed to find the recall for BF is the same as that of ISMBGT, as mentioned in Section 3.7.4.2. In other words, the correct channel of the targeted song is the channel with the highest similarity score among all channels of that song.

In Figures 3.5 - 3.9 we present the recall vs. the top-$5, 10, 20, 50, 100$ returned results for the synthetic queries belonging to each of the 5 intervals for both BF and ISMBGT, when varying the number of the k best reference sequences used in the filter step and the sampling parameter s (for ISMBGT). At the upper part of these figures we present the recall vs. top-K for different values of k, when a) there is no sampling during the filter step (left sub-figure), i.e., $s = 1$, and b) the sampling parameter is $s = 3$ (right sub-figure). At the bottom part of these figures we show how the recall vs. top-K varies for different values of s, for the value(s) of k that seemed more promising when we applied sampling with $s = 3$. The same type of information is shown in Figures 3.10 - 3.13 for the hummed queries of each of the 4 intervals. The results when $s = 1$ and $s = 3$ are now merged in one sub-figure, and the results when varying s for one or two values of k are also shown. In all figures, for ease of readability, ISBMGT is denoted as EMB. It is clear that high recall values are desirable for small K, while s and k are as high and small, respectively, as possible. Next, we present for each interval the combination of k and s values that attained the highest recall curve while still achieving significant speedup compared to BF.

Starting with the synthetic queries and b_1, BF achieves 64% recall for the top-5 returned results (16 correct results out of 25 queries) and 84% for $K = 10, 20, 50$ (21 correct results), while ISMBGT for $k = 3$ and $s = 5$ achieves 72% recall for the top-$5, 10, 20, 50$ returned results (Figure 3.5). This shows that by even retrieving as few as the top-5 returned results the indexing method achieves higher recall than the BF, which is ideal. For b_2 BF has a recall of 97.77% (88 correct results out of 90 queries) for $K = 5, 10$ and 98.88% for $K = 50$, while ISMBGT gives 84.44% when $k = 20$, $s = 3$ and $K = 5$ (76 correct results), and 85.55%, 86.66%, and 90% for $K = 10, 20, 50$, respectively (Figure 3.6). This difference of approximately 10% is perfectly acceptable, especially if we consider the runtime speedup and efficiency

evaluation measure when compared to BF, as shown in the next sections and Tables 3.4 and 3.7. Similar conclusions hold for intervals b_3, b_4 and b_5. More specifically, BF has 100% recall for all of these intervals when $K = 5$. Regarding ISMBGT, for b_3 with $k = 18$ and $s = 3$ it achieves 87.77% recall for $K = 5$ (158 correct results out of 180 queries). For b_4 with $k = 20$, $s = 5$, and $K = 5$ its recall is 88.15% (119 correct results out of 135 queries) while for 50 returned results it slightly increases to 88.88%. With respect to the bucket of largest query lengths, i.e., b_5, ISMBGT with $k = 10$, $s = 3$, and $K = 5$ has 85.71% recall and 87.14% for $K = 20$ (61 correct answers out of 70 queries). For more details on the different values of parameters and recalls for b_3, b_4, and b_5, please refer to Figures 3.7, 3.8, and 3.9.

With regard to the hummed queries, we observe the following accuracies for the different intervals. The query songs of the first interval, b_1, are very hard to identify even by BF, since for $K = 5$ there are only 3 correct answers, while for $K = 10, 20,$ and 50 there are 5, 6, and 8 correct answers, respectively. As happens with BF, ISMBGT with $k = 7$, $s = 3$ and $K = 5$ has 3 correct results, while for $K = 10, 20,$ and 50 it has 4, 5, and 7, respectively (Figure 3.10). According to these recall values, we can clearly see that although the query lengths are very small for interval b_1, ISMBGT is almost identical in accuracy with BF, and that the number of reference sequences needed in the filter step is relatively low ($k = 7$). Referring to the sampling parameter s, it cannot be greater than 3, since the small query lengths do not allow to skip more candidate endpoints during the filter step and still attain good retrieval accuracy. Similar to the first interval, BF for b_2 attains 6 out of 30 queries to have their correct answer in the top-5 returned results, while ISMBGT with $k = 2$ and $s = 7$ has 7 correct results (for $K = 5$), i.e., even higher than BF. For $K = 10$ and 20 there are 9 and 14 correct results for BF, while for ISMBGT we get 7 and 8 (Figure 3.11). The reason for not getting higher recall for the indexing

framework is that only 2 reference sequences are used and the sampling parameter is very high for a real QBH scenario. Real queries of b_3 are again very hard to identify. BF has 22 queries (out of 39) with their targeted song in the top-5 returned results, while ISMBGT has 15 with k being only 1 and $s = 7$ (Figure 3.12). This combination of values for k and s is considered to be extremely good for our proposed indexing framework and the QBH application, and thus different parameters can be used in order to increase the recall (but in parallel decrease the speedup). For the queries that belong to the final bucket, b_4, BF decides correctly for 6 (out of 16) queries when $K = 5$, while ISMBGT for more queries, i.e., 7, with $k = 10$ and $s = 5$. When $K = 10$ and 20 BF has 9 and 11 correct results, and ISMBGT has 8 for both values of K (Figure 3.13).

3.7.4.4 Runtime

In Table 3.4 the runtimes of BF and ISMBGT are shown for the synthetic and hummed queries (suffix '-S' and '-H', respectively) for each query length interval (b_1-b_5). The runtimes presented for ISMBGT correspond to the best recall that was achieved, as described in Section 3.7.4.3 and shown in Figures 3.5 - 3.13, and the values of k and s are given in parentheses. Furthermore, the *ratio* of brute-force SMBGT runtime to that of ISMBGT per bucket is provided. Ratio is essentially the indicator of how well ISMBGT performs in terms of speedup over the brute-force approach. In addition, although no sampling gives better accuracies than applying it, in Table 3.4 we show the runtimes (and ratios) for the values of s and k that yield the highest recall (for each b_i, $i = 1 - 5$), when $s > 1$. This is because, as mentioned in Section 3.7.4.3, we want to maximize recall while gaining as much as possible with regard to runtime when comparing ISMBGT to BF.

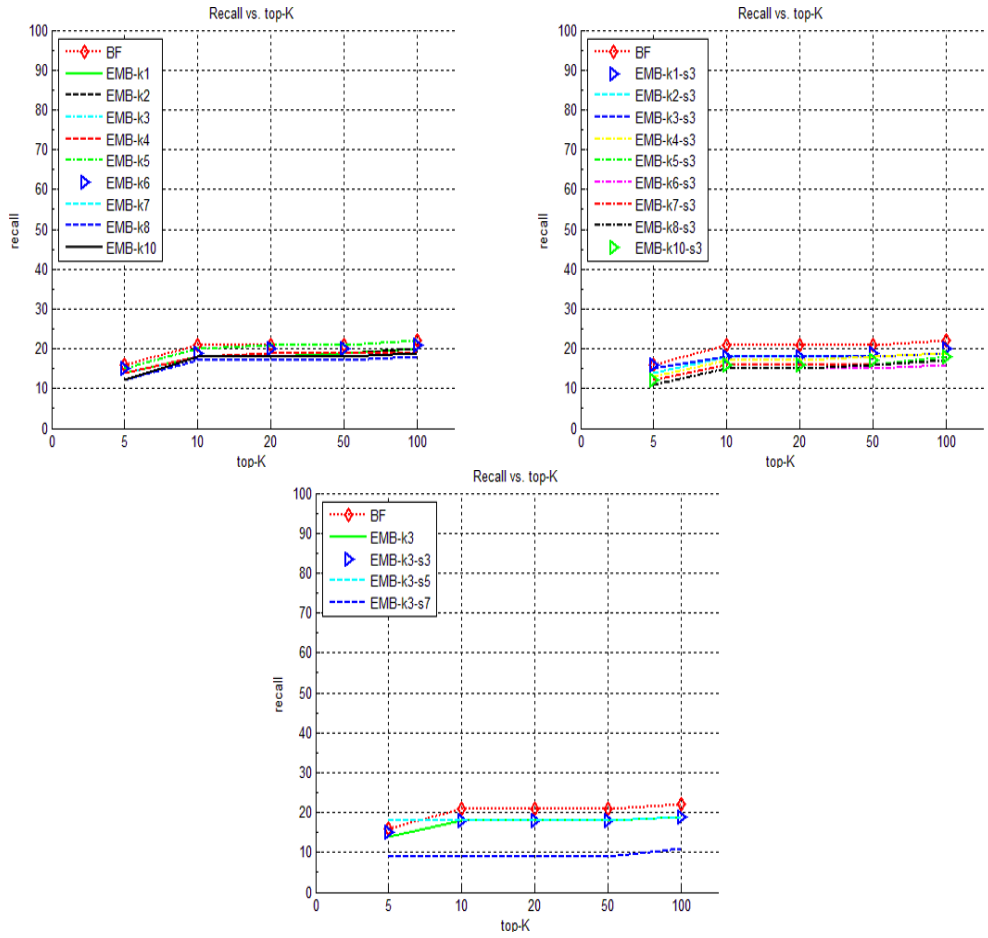


Figure 3.5: Recall for the synthetic queries of bucket b_1. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (up-left) $k = 1-8$ and 10 with no sampling ($s = 1$); (up-right) $k = 1-8, 10$ with $s = 3$; (bottom) $k = 3$ with $s = 1, 3, 5, 7$.

By looking at Table 3.4, it is clear that the ratio is much greater for the hummed queries than the synthetic ones for every query length interval. To be more specific, the ratios for the synthetic queries for $b_i, i = 1-5$, are 1.99, 2.25, 3.2, 4.28, and 5.85, while for the hummed queries they are 2.72, 7.82, 4.47, and 8.32 (no hummed queries for b_5). The main reason for getting better ratios for the hummed queries per interval compared to those of synthetic queries (or else for gaining more in terms of runtime for the hummed queries than for the synthetic) is that the BF-H runtimes are on

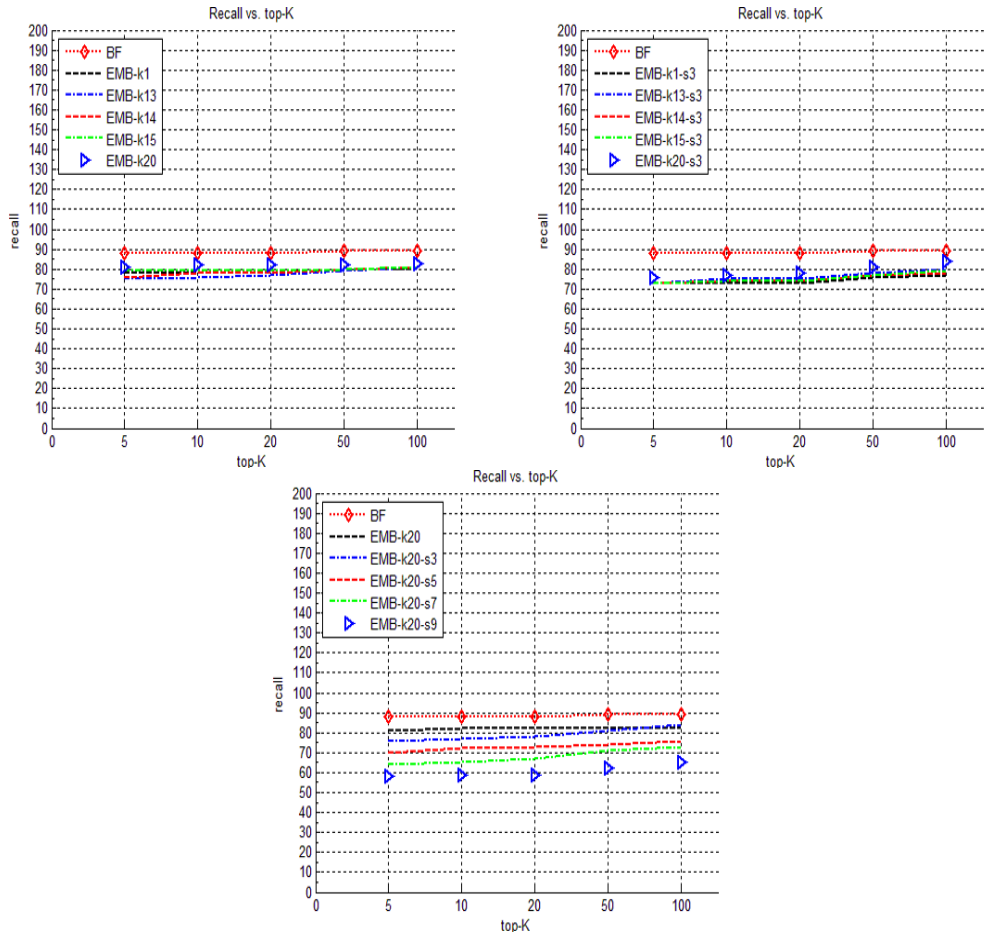


Figure 3.6: Recall for the synthetic queries of bucket b_2. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (up-left) $k = 1, 13 - 15$ and 20 with no sampling ($s = 1$); (up-right) $k = 1, 13 - 15, 20$ with $s = 3$; (bottom) $k = 20$ with $s = 1, 3, 5, 7, 9$.

average 2 times higher than those of BF-S. This happens due to the fact that the computation scheme of SMBGT is more complex for the hummed queries than for the synthetic ones. There is no tolerance when applying SMBGT to the synthetic queries, while for the hummed queries variable absolute tolerance with $t = 0.2$ is used, which by default is a more sophisticated tolerance type. What is more, ISMBGT runtimes do not differ much between 'S' and 'H'. In fact, for no interval is ISMBGT-H runtime greater than 1.67 times the ISMBGT-S runtime. In contrast, ISMBGT-H runtime

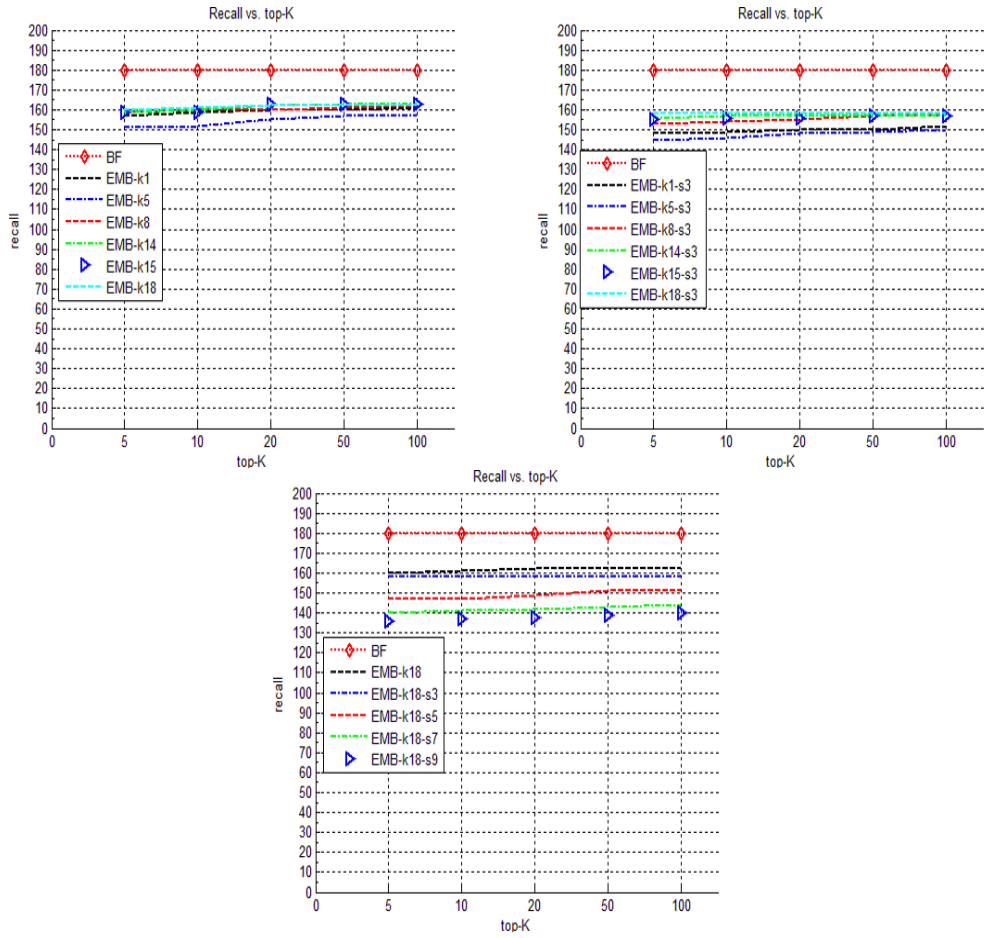


Figure 3.7: Recall for the synthetic queries of bucket b_3. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (up-left) $k = 1, 5, 8, 14, 15, 18$ with no sampling ($s = 1$); (up-right) $k = 1, 5, 8, 14, 15, 18$ with $s = 3$; (bottom) $k = 18$ with $s = 1, 3, 5, 7, 9$.

for b_2 is even smaller than ISMBGT-S runtime by 1.63 times. The reason for this decrease is that in the latter case (ISMBGT-S) $k = 20$ and $s = 3$, while in the former case (ISMBGT-H) k is an order of magnitude smaller and the sampling parameter is more than 2 times greater, i.e., $k = 2$ and $s = 7$. This combination of parameters leads to computing distances between much smaller and fewer vectors during the filter step.

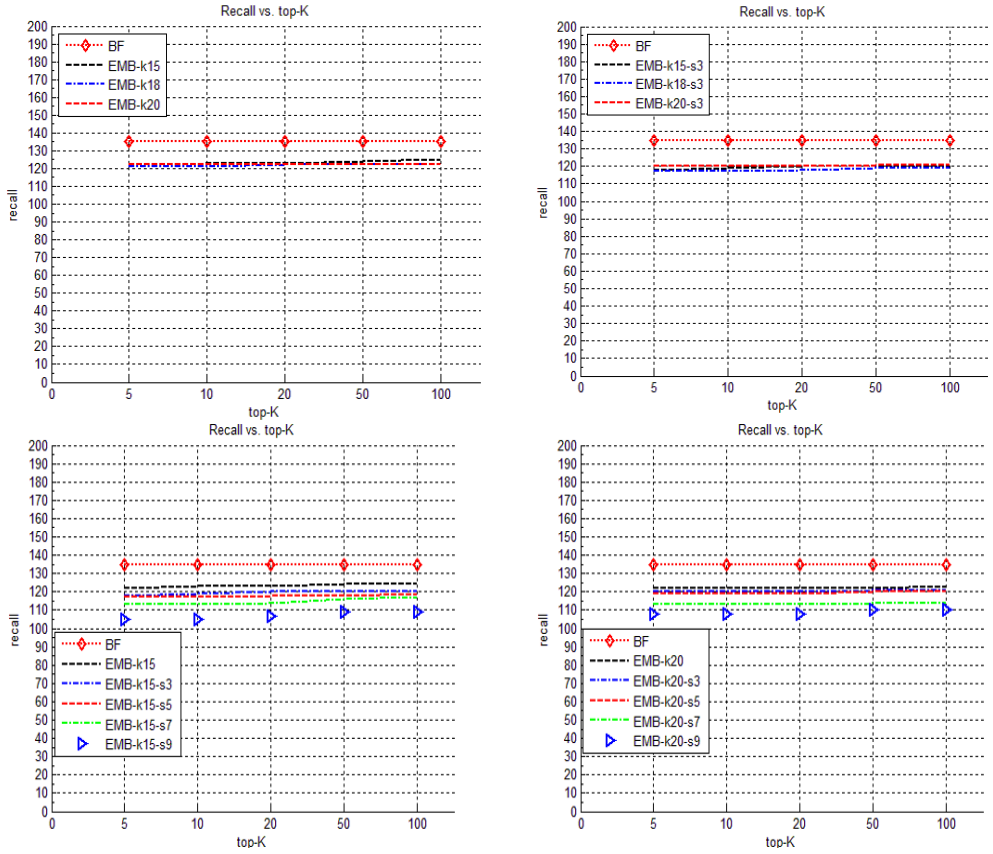


Figure 3.8: Recall for the synthetic queries of bucket b_4. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (up-left) $k = 15, 18, 20$ with no sampling ($s = 1$); (up-right) $k = 15, 18, 20$ with $s = 3$; (bottom-left) $k = 15$ with $s = 1, 3, 5, 7, 9$; (bottom-right) $k = 20$ with $s = 1, 3, 5, 7, 9$.

Another important observation is that the ratio values increase as we move from b_1 to b_5 for both synthetic and hummed queries. The main reason underlying this is that the runtimes of BF-S and BF-H increase as well, but more than the runtimes of ISMBGT-S and ISMBGT-H. The complexity of ISMBGT depends on the length of the queries and the length of the segments of the database channels that are evaluated during the refine step. These segments get bigger as we move to buckets of greater values because they are created based on r, which in turn is defined as a factor of $|Q|$. We have to note that the ratio for the hummed queries for b_2 is 7.82, which is

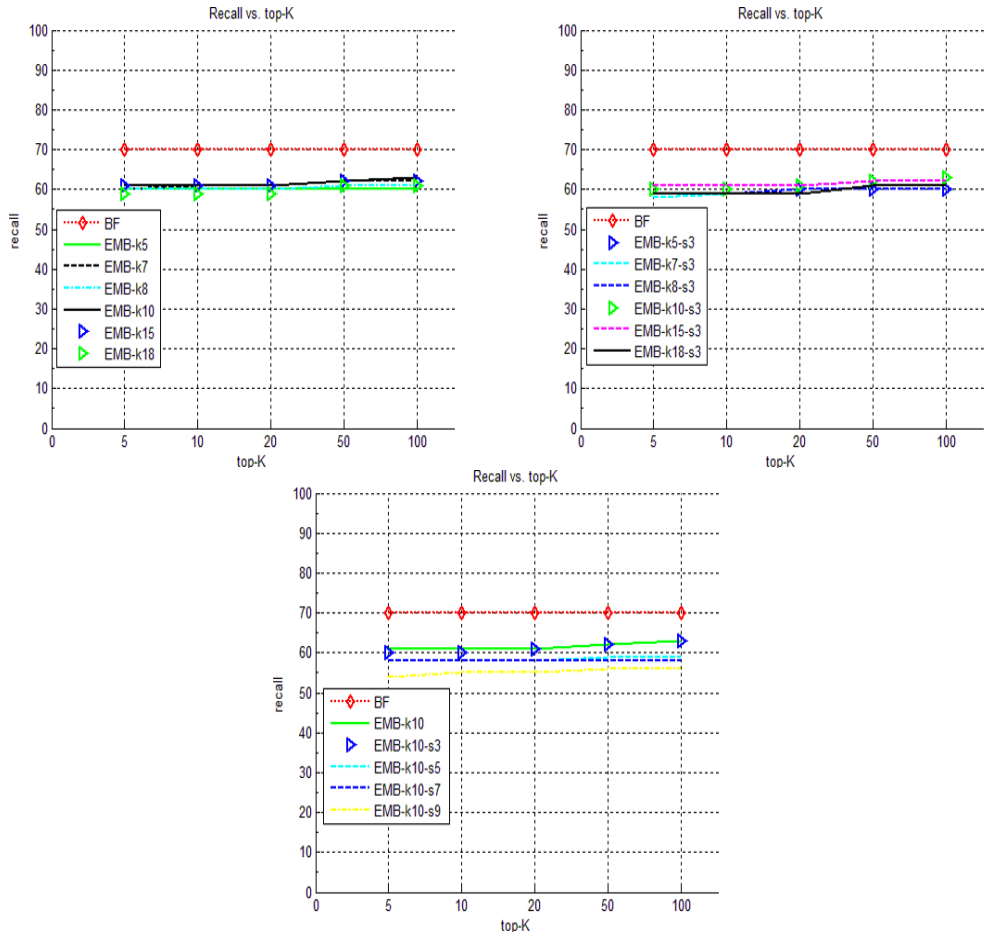


Figure 3.9: Recall for the synthetic queries of bucket b_5. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (left) $k = 5, 7, 8, 10, 15, 18$ with no sampling ($s = 1$); (right) $k = 5, 7, 8, 10, 15, 18$ with $s = 3$; (bottom) $k = 10$ with $s = 1, 3, 5, 7, 9$.

greater than the 4.47 ratio for b_3. Although $s = 7$ for both query length intervals and $k = 2$ for b_2 while it is $k = 1$ for b_3, the difference in the query lengths between those intervals makes ISMBGT-H yielding much lower total runtime for b_2 (3.68 seconds) than for b_3 (8.71 seconds).

The average runtimes for each step of ISMBGT (i.e., embedding, filter, and refine) per interval for the synthetic and hummed queries are given in Tables 3.5 and 3.6, respectively. The times presented correspond to the k and s values shown in

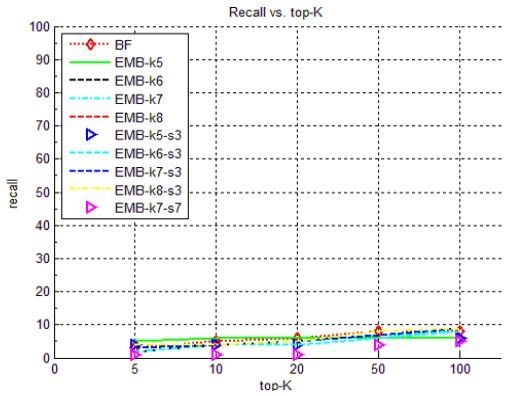


Figure 3.10: Recall for the hummed queries of bucket b_1. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best $k = 5 - 8$ reference sequences with sampling parameter $s = 1, 3$, and $k = 7$ with $s = 7$.

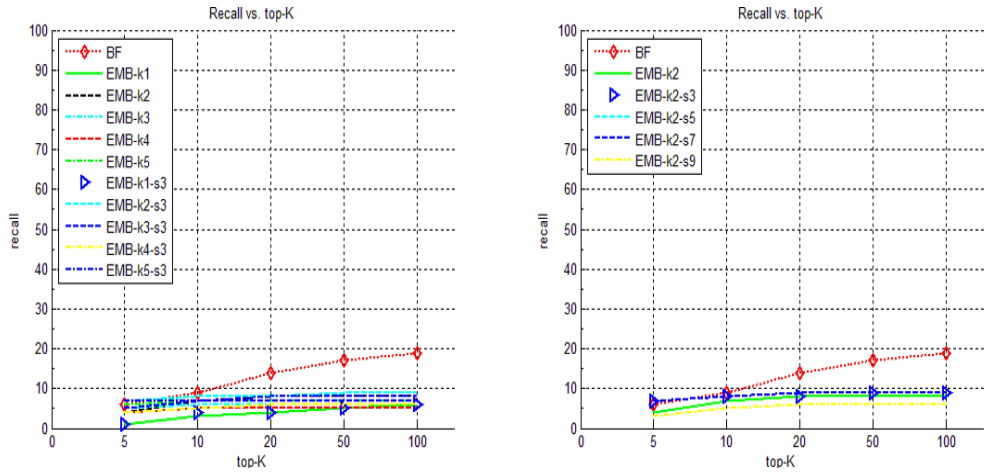


Figure 3.11: Recall for the hummed queries of bucket b_2. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (left) $k = 1-5$ with $s = 1, 3$; (right) $k = 2$ with $s = 1, 3, 5, 7, 9$.

parentheses in Table 3.4, i.e., the ones that lead to the highest accuracies for each query length interval while saving the maximum possible computational time.

We observe that the embedding step times increase per interval for both synthetic and hummed queries. This is reasonable because the only difference among intervals is the query length, which increases, and the time for this step depends on the complexity of SMBGT that is influenced by $|Q|$ and $|R|$ ($|R|$ also increases

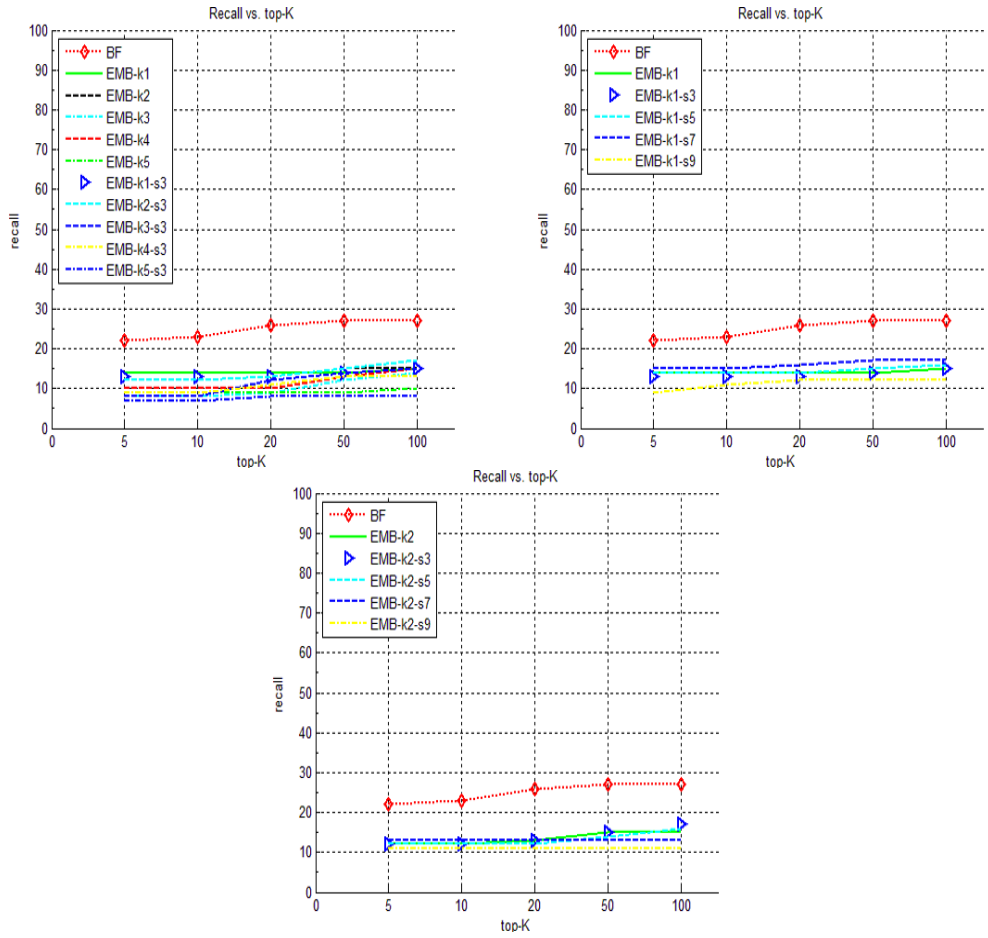


Figure 3.12: Recall for the hummed queries of bucket b_3. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (left) $k = 1-5$ with $s = 1, 3$; (right) $k = 1$ with $s = 1, 3, 5, 7, 9$; (bottom) $k = 2$ with $s = 1, 3, 5, 7, 9$.

as shown in Table 3.3). The maximum value for synthetic and hummed queries is achieved for b_5 and b_4 and is only 0.15 and 0.21 seconds, respectively.

Regarding the filter step times, they depend on the values of k and s. For synthetic queries and b_1 the runtime is only 0.58 seconds. This is due to the fact that k is only 3 and s is relatively high, i.e., $s = 5$. For b_2, b_3, and b_4 the filter times are much greater, i.e., 2.76, 2.55, and 2.13 seconds, since the number of reference sequences used is 18, 20, and 18, respectively. For b_4 the filter time is smaller than

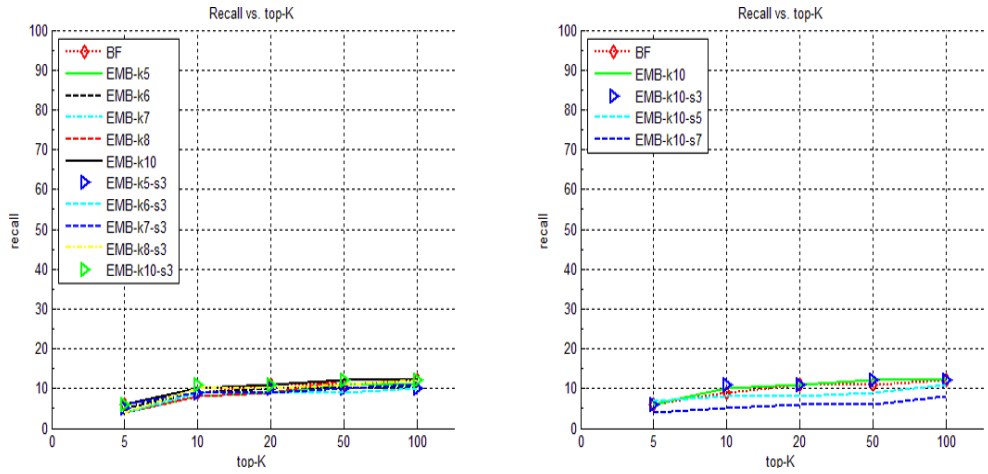


Figure 3.13: Recall for the hummed queries of bucket b_4. Recall is shown in absolute numbers for BF and ISMBGT for $perc = 1\%$ and the best k reference sequences and sampling parameter s: (left) $k = 5 - 8, 10$ with $s = 1, 3$; (right) $k = 10$ with $s = 1, 3, 5, 7$.

that of b_2 and b_3 because of using a greater sampling parameter s (5 instead of 3). This indicates that even though $|Q|$ and $|R|$ are higher for b_4, sampling plays a significant role in the speedup of ISMBGT. Filter step runtime for b_5 is between the aforementioned times (1.63 seconds), mainly because of the value of k, which is 10, i.e., in-between the k value of the other intervals. For the hummed queries the smallest runtimes are for b_2 (0.36 seconds) and b_3 (0.3 seconds), because k is only 2 and 1, respectively. Since $s = 7$ for both buckets, and they only differ in the value of k, the time is greater for b_2. Also, the higher value of k, which is 7, for b_1 leads to a filter time that is just over 2 seconds. What is more, although k is greater for b_4 than b_1, b_4 has a smaller filter time since its sampling parameter is 5 (for b_1 $s = 3$). The maximum value for the synthetic queries is 2.76 (bucket b_2) and for the hummed queries 2.04 seconds (bucket b_1), which confirm the importance of having a fast filter step in this embedding-based indexing approach.

For the refine step of ISMBGT and the synthetic queries we can see that the time increases when we examine intervals of greater query lengths. This is because the runtime of the refine step depends on $|Q|$ and the lengths of the segments of channels to which SMBGT is applied, while the number of these channels does not deviate much. Regarding the hummed queries there is no trend in the runtimes, verifying here the dependence of the refine step on the number of channels to which SMBGT is applied. Finally, the maximum time for the refine step and the synthetic queries is 5.93 seconds (bucket b_5), while for the hummed queries it is 8.36 (bucket b_3), showing that this step is the one that essentially determines the total runtime of ISMBGT.

Table 3.4: Runtimes of brute-force SMBGT and ISMBGT for the synthetic (BF-S and ISMBGT-S) and the hummed queries (BF-H and ISMBGT-H) per query length interval. *Ratio* is the ratio of times of BF to ISMBGT. The two numbers in parentheses correspond to the best k reference sequences selected in the filter step and the sampling parameter s used.

	Time (in seconds)				
Method	b_1	b_2	b_3	b_4	b_5
BF-S	7.1301	13.4625	20.9028	29.3055	45.0879
ISMBGT-S	3.5777 (3, 5)	5.9902 (20, 3)	6.5189 (18, 3)	6.8446 (20, 5)	7.7136 (10, 3)
Ratio	1.9929	2.2474	3.2065	4.2816	5.8452
BF-H	16.3179	28.763	38.9623	62.1828	-
ISMBGT-H	5.9953 (7, 3)	3.676 (2, 7)	8.7088 (1, 7)	7.4714 (10, 5)	-
Ratio	2.7218	7.8245	4.4739	8.3228	-

Table 3.5: Average times of ISMBGT for synthetic queries. The values of k and s per query length interval are given in Table 3.4.

	Query Length Intervals				
ISMBGT Steps	b_1	b_2	b_3	b_4	b_5
Embedding	0.0088	0.0164	0.0294	0.0635	0.1503
Filter	0.5812	2.7607	2.5453	2.1275	1.6274
Refine	2.9709	3.2131	3.9442	4.6536	5.9309

Table 3.6: Average times of ISMBGT for hummed queries. The values of k and s per query length interval are given in Table 3.4.

	Query Length Intervals			
ISMBGT Steps	b_1	b_2	b_3	b_4
Embedding	0.0080	0.0190	0.0456	0.2088
Filter	2.0427	0.3643	0.2990	1.1981
Refine	3.9447	3.2913	8.3647	6.0607

3.7.4.5 Efficiency

In Tables 3.7 and 3.8 we present the *min*, *max*, *mean*, and *median* statistics for the efficiency evaluation measure of ISMBGT per query length interval for the synthetic and hummed queries, respectively. All statistics correspond to the values of k and s shown in Table 3.4 in parentheses, and certainly depend on the number of channels evaluated during the refine step. As a result, we cannot compare the efficiency values among different intervals.

Starting with the synthetic queries (Table 3.7), we observe that the min values for all intervals, i.e., the minimum efficiency achieved by any query for each bucket, range approximately from 2.61% (bucket b_3) to 3.36% (bucket b_1). The max values range from 9.59% (bucket b_2) to 18.62% (bucket b_5), showing that in the worst case for the synthetic queries we visit less than one fifth of the database and still manage to achieve high recall. Moreover, the mean and the median are very close to each other for all intervals, except for b_1 as the mean is affected by the max value. In addition, the median (and the mean) increase when moving from lower to higher query length intervals, i.e., from b_1 to b_5, except for b_3 where the efficiency is a little higher than that of b_4. More specifically, the mean efficiencies per interval are the following: 4.95%, 6.18%, 8.28%, 7.54%, and 10.71%. One reason for this is the fact that, for

the intervals that involve larger queries, the SMBGT has to be evaluated for larger queries and segments of the database, while the number of channels evaluated does not deviate much among the intervals. The aforementioned efficiency values show that there is a significant speedup of ISMBGT compared to BF as (approximately) at most only 10% of the database is actually evaluated with SMBGT.

Regarding the hummed queries, the min values for all intervals range approximately from 3.8% (bucket b_2) to 6.77% (bucket b_3), while the max values range from 6.74% (bucket b_1) to 40.84% (bucket b_3). Also, as happens with the synthetic queries, mean and median are very similar per bucket. In particular, the median efficiencies for the four query length intervals are the following: 5.75%, 4.43%, 20.49%, and 7.76%. These values are very low for b_1, b_2 and b_4, showing again the valuable trade-off between retrieval runtime and recall that we can get (Figures 3.10, 3.11, 3.13). For b_3, though, efficiency is 20.49%. This is because of the nature of the hummed queries that belong to that interval; many more different channels are evaluated, and consequently more segments are evaluated with SMBGT. We note, though, that visiting on average one fifth of the database for queries of lengths 32 up to 49 still shows the superiority of the proposed indexing scheme over the brute-force search.

Table 3.7: Efficiency of ISMBGT for synthetic queries. The values of k and s per query length interval are given in Table 3.4.

	Query Length Intervals				
% of |DB| visited	b_1	b_2	b_3	b_4	b_5
Min	3.3583	3.0687	2.6058	3.2933	2.7061
Max	13.8881	9.5921	17.5954	11.1619	18.6247
Mean	6.2597	6.2138	8.4114	7.5427	10.8905
Median	4.9476	6.1826	8.2846	7.5408	10.7105

Table 3.8: Efficiency of ISMBGT for hummed queries. The values of k and s per query length interval are given in Table 3.4.

	Query Length Intervals			
% of |DB| visited	b_1	b_2	b_3	b_4
Min	5.1977	3.8012	6.7706	6.0090
Max	6.7417	9.3819	40.8400	9.3077
Mean	5.9357	5.2568	20.8187	7.5871
Median	5.7459	4.4315	20.4938	7.7620

3.7.5 Highlights

Based on the results shown in this section, we can first conclude that SMBGT and SMGT are highly suitable for the QBH application. Regarding the synthetic queries, even for high noise levels (e.g., 50%), the recall and MRR values of both methods for the top-20 returned results were close to 100% and 1, respectively. The same was observed with the Edit distance. However, for the hummed queries Edit distance had 0% recall for $K = 10$. In contrast, SMBGT and SMGT achieved 57% and 42% recall, respectively, while for $K = 50$ they were more than 30 times higher in recall than Edit distance.

Regarding the proposed indexing approach ISMBGT, we showed that the sampling parameter s can provide significant speedups for the filter step. Specifically, ISMBGT can achieve recall values very close, or even higher for some cases, to those of BF by using relatively large sampling rates of up to $s = 7$, hence considering only $1/7^{th}$ of the embedding of the whole database.

We should also note that, although all the obtained results are based on randomly selecting reference sequences for all intervals, ISMBGT can still achieve high recall values. Consequently, applying a more sophisticated method for selecting reference sequences can lead to even better accuracies, confirming that ISMBGT is meaningful and can be successfully applied to other time series application domains.

Moreover, by looking at the values of the ratio of brute-force SMBGT runtime to that of ISMBGT for synthetic and hummed queries, we reached the following conclusions. For the synthetic queries the minimum gain we can achieve with ISMBGT in terms of runtime is 2 times (bucket b_1), while the maximum gain is 5.85 times (bucket b_5). Also, the longer the query is the better the runtime gain. Regarding the hummed queries, the minimum gain is 2.72 (bucket b_1) and the maximum 8.32 (bucket b_4). Again, the longer the query the better the retrieval cost (or greater ratio); for queries of length in $[33, 53)$ significant speedup can also be achieved (7.82). These ratios show the importance of ISMBGT, especially in real-time scenarios.

Furthermore, comparing the runtimes of the three steps of ISMBGT we can conclude that the most demanding one is the refine step. More specifically, the maximum time required for the embedding step is 0.15 seconds for the synthetic queries (bucket b_5) and 0.21 seconds for the hummed queries (bucket b_4). For the filter step the time ranges from 0.58 to 2.76 seconds for the synthetic queries, and from 0.36 to 2.04 seconds for the hummed queries. The refine step requires at most 5.93 seconds for the synthetic queries (bucket b_5) and 6.06 seconds for the hummed ones (bucket b_4).

Finally, with regard to efficiency (for ISMBGT), its median value is 4.94% (bucket b_1) and increases up to at most 10.71% (bucket b_5) for the synthetic queries. For the hummed queries, the lowest median value is 4.43% (bucket b_2), while the largest one is 20.49% (bucket b_3). These results indicate that, in the worst case, only one fifth of the whole database will be (on average) actually evaluated using the costly SMBGT method, making the proposed indexing method highly efficient for the QBH application domain.

3.8 Conclusions and Future Work

In this chapter, we presented a subsequence matching framework suitable for QBH, and the SMBGT method, which is based on this framework. In addition, we presented an embedding-based subsequence matching indexing approach under SMBGT, which allows for gaps in both query and target sequences, variable tolerance in the matching of elements, and constrains the maximum match length and optionally the minimum number of matched elements. The brute-force approach was shown to outperform several DP-based subsequence matching methods and a model-based probabilistic method in terms of accuracy, after extensive experimental evaluation on a large music database using hummed and synthetic queries. We also showed that the filter-and-refine indexing method can achieve speedups of up to an order of magnitude compared to brute-force SMBGT. A very important remark is that ISMBGT achieves high recall values, which are very close to those of brute-force SMBGT, even by using random reference sequences that are tailored for the given query. This is a substantial advantage over existing embedding-based methods for subsequence matching.

Directions for future work include testing the performance of ISMBGT in other application domains where noise is present. Additionally, we could investigate further ways of compressing the embedding space, for example, by applying commonly used time series segmentation methods [101].

CHAPTER 4

HUM-A-SONG: A SUBSEQUENCE MATCHING WITH GAPS-RANGE-TOLERANCES QUERY-BY-HUMMING SYSTEM

In this chapter, we present the "Hum-a-song" QBH system [102], which allows the user to record a melody that she recalls and search a large music database to retrieve the top-K most similar songs to the sung melody. The system gives the opportunity to the user to select among various similarity or distance based methods to assess the similarity between the query and the database songs. Secondly, since QBH is a very noisy domain, we are interested in performing robust and efficient subsequence matching. Consequently, apart from the fact that the user is given the opportunity to test and compare several methods, the SMBGT method is also implemented and included in our system. Furthermore, our system is open source, so that it can be extended to include, for example, more similarity/distance measures for comparison and benchmarking purposes. Last but not least, since we approach the QBH problem by mapping it to the time series domain, the system can be easily extended for other time series application domains in which subsequence matching would be of interest, such as finding specific patterns in sensor, financial, and weather data among others.

4.1 Related Work

Several commercial products exist for music retrieval and QBH. Given a part of a song recording Shazam (http://www.shazam.com) identifies the song. The limitation of Shazam is that it is not applicable to QBH, i.e., it only works for queries

that are recordings of songs (with potential noise) having an exact match in the target database and not for songs that are hummed by some individual. In addition, Soundhound (known as Midomi until 2009, http://www.soundhound.com), is an application that works for QBH, as opposed to Shazam. However, it is not open-source and the technical details of its matching methods are unknown.

4.2 Hum-a-song - DEMO

4.2.1 Recording Queries

The user of the system is first asked to record the part of the song she recalls and is interested in finding in the database. For this purpose, the *Akoff music composer-version 2.0* [1], a tool commonly used for evaluating QBH systems is used [61]. The user is asked to hum the part of song close to a microphone, and to avoid singing with lyrics. Then, the melody sang is converted to MIDI through Akoff. Finally, to get a 2-dimensional time series from the melody, the pitch at every time click is extracted, and tuples ⟨pitch, click⟩ are converted to representation 5 of Table 2.1. It is noted that the user can also perform an experiment on a pre-recorded hummed query.

After recording a query song, the user is able to select among a variety of 2-dimensional time series representations. For the purposes of our demo, and since we are interested in note transitions, so as not to check all possible transpositions of a melody nor to scale in time when comparing time series, we consider the following encoding schemes: ⟨pitch interval, IOIR⟩ and ⟨pitch interval, LogIOIR⟩.

[1] http://www.akoff.com/music-composer.html

4.2.2 Method Selection

Since we are interested in evaluating a QBH system, apart from the SMBGT and SMGT methods described in Chapter 3 [79], the user is allowed to choose among a variety of methods that can be applied to music retrieval. These methods, performing DP-based subsequence matching, are SPRING [67], a version of Edit distance suitable for music retrieval [66], and two DTW-based [62, 63] (denoted DTW_s and DTW_c). We note that Edit has been modified to account for LogIOIR and quantizations (Table 2.1), and SPRING allows for varying r [79]. These DP-based methods were also described in Section 3.4.

4.2.3 Tuning Parameters and Tolerances

Having selected the query, the representation, and the method, the user has to provide the appropriate parameters for the requested method. For SMBGT, the number of consecutive gaps allowed in both X and Q has to be given, which are identified by the α and β positive integers, respectively. Also, parameter r is used to eliminate large matches, e.g., $r = 1.2 * |Q|$, and δ to bound the minimum number of matching elements, e.g., $0.5 * |Q|$. As mentioned in the previous chapters, since in QBH it is quite usual for users to make instant humming errors [72] error-tolerant matches should be allowed. The "Hum-a-song" system provides the possibility of applying constant or variable absolute or relative tolerances, as described in Section 3.3. If the user selects SMGT then the input parameters are the same, except that no constraint is imposed on the lengths of the allowed gaps. With regard to SPRING, since it has been modified not to account for infinite matching lengths, the user has to provide r, as happens with SMBGT and SMGT. Finally, DTW_s and DTW_c require

the initialization parameter c, while the version of Edit distance used [79] does not require any input parameter.

4.2.4 Evaluation

Below, we present the final steps for getting the final result set for the hummed query. However, for completeness and demo purposes, our system has some pre-inserted synthetic and hummed query sets (that the user may use to compare accuracies between different methods), instead of humming a query.

4.2.4.1 Database

The music database that we used for our system (and its transformation to 2-dimensional time series) is the same as the one described in Section 3.7.1. We note that the 40,891 time series are loaded in memory at runtime.

4.2.4.2 Hummed Query

The user also has to insert the number of top-K songs she wishes to be returned in the result set for the melody sung. The top-K results appear in the final screen of our system, along with the 2-dimensional time series plot of the given melody with the top candidate and the time for returning the results. What is more, the user can select and listen to a song appearing in the list of results, listen to the query, see more details regarding the selected song, i.e., similarity/distance score, channel giving that score, start and end points of the matching, and also the 2-dimensional plot of the song's channel giving the best score. The performance of the system depends on the hummed query, the method, and the parameters given. The whole procedure can be certainly repeated by recording another melody (*Main Menu* button). An example

of setting an experiment for our system along with the experimental results is shown in Figure 4.1.

4.2.4.3 Testbeds

Another possibility offered by the proposed system is the testbeds, comprising five synthetic query sets and one hummed set. These query sets were described in detail in Sections 3.7.1.2 and 3.7.1.3.

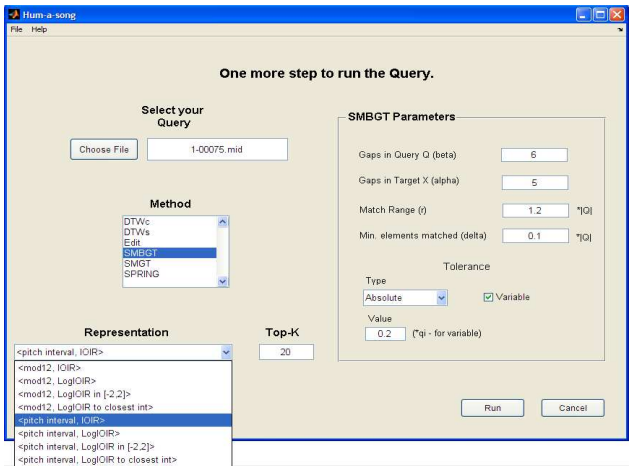
When choosing a testbed, in the final screen, apart from the time required for the results to be returned, the user is presented with three evaluation measures for the method and representation selected: *recall*, *Mean Reciprocal Rank (MRR)* [100], and the *average rank (AR)* of all queries in the selected query set. It has to be mentioned that all measures are of particular importance for evaluating a QBH method, since recall indicates if the method identifies the correct answer in the top-K results, and MRR along with average rank show if recall can be improved by decreasing K. The representation, number of top-K results used, and the parameters of the selected method giving the best recall are also shown.

Our system is implemented in Matlab, currently runs in Windows Operating System, and can be downloaded from http://vlm1.uta.edu/∼akotsif/hum-a-song/.

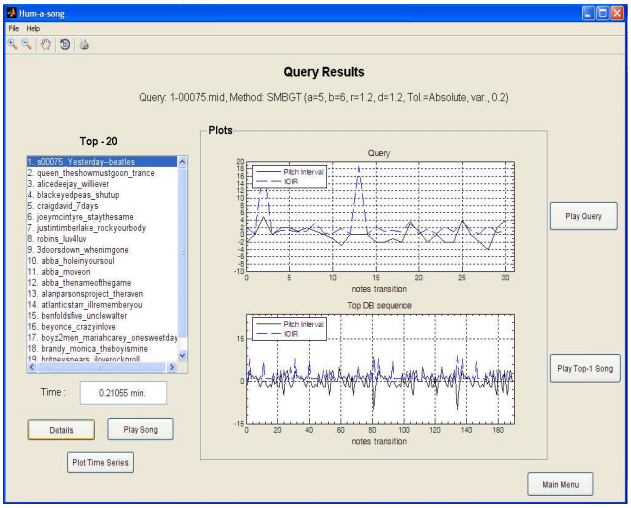
4.3 Conclusions

Motivated by the QBH application, we have developed a demo system named "Hum-a-song" that allows the user to hum a part of a song that she recalls and find the most similar songs to this. This is done by transforming this problem to the time series domain, where the query and the songs of the music database are mapped to 2-dimensional time series and then applying a matching method to get the subsequences

of the database that best match the query. Several methods have been implemented and given as an option, including the robust and efficient SMBGT method. Finally, our system can be easily extended to other time series application domains, basically due to the fact that the queries and the target sequences are represented as time series and the methods employed perform on top of such representation.



(a) Setting the Query.



(b) Query results.

Figure 4.1: "Hum-a-song": Screenshots of setting the experiment and getting the results for SMBGT, where $|Q| = 31$, the database has 4,000 sequences, and the target "Yesterday" song is the Top-1. For the target sequence, $Score = 20$, $Startpoint = 28$, $Endpoint = 56$.

CHAPTER 5

GENRE CLASSIFICATION OF SYMBOLIC MUSIC WITH SMBGT

Music can be seen as an effective means of stimulating and focusing attention. Particularly for specific groups of people who are weak in responding to other physical interventions, music can be highly significant. Hence, music can be considered as an assistive technology with high therapeutic and educational functionality for children and adults with disabilities. Skill areas such as mental retardation, autism, and learning disabilities can be affected by music therapy. Music can be used to structure an entire therapeutic intervention in order to maintain attention. It can be used as means of alerting people for important information or interactions, or it can function as a calming down mechanism when anxiety intervenes with cognitive focus. In addition, music is of high importance in learning, as it can provide significant assistance in memorization.

A problem of particular interest to the community of music informatics as well as to the communities of data mining and assistive technologies is that of classifying a query song of unknown category to one of several existing and known *categories/genres*.

Music genre classification is highly related to the area of assistive environments, since it can facilitate many of the music functionalities described above. For example, music systems with therapeutic and educational functionalities need to be adapted based on the music preferences of the end-users, such as children, patients, or disabled. Hence, being able to identify the genres of songs that fit to each category of end-users is imperative in such settings as it can assist effectively in medical treatments. As

another example, consider the task of learning and memorization. It would be very useful to identify for each individual the music genres that are more suitable when performing a learning task or are less disruptive than other genres.

Similarity search is the basis for a variety of applications, such as playlist generation, similarity-based browsing interfaces, and recommendation systems. It has to be mentioned though that end-users are more likely to browse and search by genre than by recommendation, artist similarity or music similarity [103]. As a result, genre classification has been studied in the music literature, and research focuses on exploiting machine learning and data mining methods for meaningful results.

As mentioned by Scaringella et al. "Music genres are categories that have arisen through a complex interplay of cultures, artists, and market forces to characterize similarities between musicians or compositions and organize music collections" [104]. Music genres can be characterized by using several types of rules [105], while there is a good work by Aucouturier and Pachet on how to represent music genres [106].

Constructing automatic music classifiers is of great interest for a variety of reasons [107]. Starting with genre classifiers, which are the focus of this chapter, they can be used to understand the important characteristics of music that help distinguishing particular genres and how these characteristics vary. What is more, classifiers can be used to categorize recordings whose authorship is unknown to composers. In addition, results produced by automatic classifiers can be combined with sociological and psychological research and give insight on the way that humans understand musical similarity and construct clusters of music pieces.

Although much research has focused on how to represent music genres and on genre classification, still there are many problematic aspects that should be addressed, as mentioned below [108]. First of all, there is not reliable ground truth, which is not only fundamental to effectively train genre classifiers, but also to evaluate genre

classification systems. Secondly, genre classification is most of the times done by artist or album and not by individual recording that is certainly inappropriate for the classification task. Moreover, most of the times no consensus can be achieved by human annotators both on the genre of songs, due to the non-existing strict boundaries between genres [109], and on the different categories that can be used to classify songs in. Consequently, it is tough to build up a taxonomy of genres that is widely acceptable [110]. In addition, most genres have fuzzy definitions that may change from source to source and the criteria to define some genres vary. Also, some genres may overlap and individual songs may be classified to more than one genre making the classification task even harder, if only one genre per recording is allowed. Apart from these difficulties, not much psychological research has been performed on how humans perform genre classification [109, 111], while a significant amount of time is required for human annotators to classify songs. It is also important to have large training sets to create meaningful models and large test sets to have a better guarantee on the performance of the classification systems. Finally, it has been shown in the literature that increasing the number of genres and the number of songs reduces the performance of classification. All the aforementioned problems that arise in automatic music genre classification make current software tools not achieving satisfyingly high accuracy, and as a result research on this field should be continued.

In order to perform genre classification systems first extract important features representing salient information from the recording, and then classify the music piece to a music genre based on the output of one or more combined classifiers that take as input these features. These features are low-level and/or high-level. Features of the first category are based on signal processing quantities and include temporal, energy, perceptual, and spectral shape features. Furthermore, melodic and harmonic content

are better described by low-level attributes than notes or chords. High-level features, which are also referred to as semantic, include tempo, rhythm, key, instrumentation, vocal style, and meter, among others. For music classification humans use high-level information, which is better provided by recordings in symbolic format, such as MIDI. It should be noted that selecting appropriate features requires knowledge of diverse domains, such as signal processing, musicology, music theory, psychology, and in the last few years text and web-mining.

5.1 Related Work

Artist and genre classification where vectors of low-level audio features (MFCCs) are mapped to points in a Euclidean anchor space has been proposed [112]. These points are vectors of posterior probabilities of membership in the anchor classes (dimensions), which are represented by neural networks. Since points can be seen as samples from a distribution characterizing a song, the song is modeled by GMMs and an approximation to KL-divergence is used to compare songs. However, due to lack of ground truth, the classification accuracy is not very satisfying. McKay et al. showed the effectiveness of using large feature sets that are combined with feature weighting systems, high-level features, and also the importance of instrumentation-based features for genre classification when performed with k-NN, neural networks and combination of several classifiers [107]. Cataltepe et al. exploit the MIDIs and their audio versions. The MIDI is turned into a string from a finite alphabet and the distance between pieces is computed by the Normalized Compression Distance. MIDIs are also converted to audio from which timbre, rhythmic, and pitch features are extracted. Finally, diverse and independent classifiers based on Linear Discriminant Classifier and k-NN are combined with weights to perform classification [113].

SVMs have also been used for genre and music classification [114, 115], as well as HMM-based methods [116]. Moreover, McKay et al. suggested that background research from musicology and psychology has to be exploited for genre classification, as should happen with cultural features and metadata mined from the Web [108]. Other very interesting approaches for genre classification can be found in the literature [117, 118, 119, 120]. Finally, surveys on audio-based music classification and annotation, and automatic genre classification have been proposed [104, 121].

5.2 Measuring Song Similarity

To perform the genre classification task based on MIDI recordings we should be able to define the similarity or distance between songs [122]. The similarity measure that we use to compare sequences is SMBGT, presented in Chapter 3 [79]. Although SMBGT was proposed motivated by the QBH application, due to the fact that it can account for several desirable properties when matching a query and a target sequence, it is a general similarity measure for sequences guaranteeing a robust and meaningful matching. In addition, it can be used for both whole sequence and subsequence matching. The scheme for computing song similarity is the following.

Assume that we would like to compare two songs, namely S_i and S_j, which are in MIDI format and consist of $|c_i|$ and $|c_j|$ channels (there are at most 16 per song). First, for each song and channel, we extract the highest pitch at every time click (*all-channels extraction* [20]). It has to be noted that in the extraction process we exclude channel 10, since it is used for drums and cannot offer any melodic information. Then, we convert the tuples $\langle$pitch, click$\rangle$ of each channel to $\langle$pitch interval, IOIR$\rangle$, resulting in $|c_i|'$ and $|c_j|'$ converted channels, yielding transposition and time invariance. After these steps, we compare all channels $|c_i|'$ of S_i with the $|c_j|'$ channels of S_j using

SMBGT, and get $|c_i|' * |c_j|'$ scores. Finally, we retrieve the maximum score s_{ij} out of all the pairwise similarity scores, and normalize it by dividing with the maximum length of the two channels of S_i and S_j that gave s_{ij}. The resulting score is assigned as the similarity between S_i and S_j. The similarity score of a song S_i to itself is considered to be 0. This is done basically to exclude the query song itself from being a neighbor in the k-NN classification task; the less the similarity score is between two songs the less similar they are.

5.3 Experiments

Next, we provide the setup for our experiments and the experimental results.

5.3.1 Experimental Setup

5.3.1.1 Data

For the purposes of our experiments we collected 100 freely available on the web MIDI songs that cover four genres of music: Classical, Blues, Rock, and Pop. In Table 5.1 there is more information for the dataset, i.e., the total number of channels it covers or else $|DB|$, the number of 2-dimensional points corresponding to these channels, and also the number of songs per genre of music.

Table 5.1: Characteristics of the dataset used in the experiments. There are 100 songs, and the total number of channels comprising these songs in their MIDI representation, along with the number of 2-dimensional points corresponding to these channels are shown. In the final column, the number of songs belonging to each of the four selected genres is presented.

# of songs	# of channels	# of points	Genres
100	821	310,798	Blues (20), Rock (31) Classical (27), Pop (22)

We have to note that during the process of selecting MIDI songs we observed that pop and disco songs have many common characteristics and acoustic similarities as genres. As a result, we picked pop songs, which we considered to be sufficient for our experiments. Moreover, each song was labeled with one genre. This is because assigning a song to more than one genres would make the classification task biased and providing higher accuracies, since the number of songs and genres is not huge.

Following the selection of the 100 songs, for each of them we first applied the all-channels extraction process, and then converted the tuples ⟨pitch, click⟩ of each channel to ⟨pitch interval, IOIR⟩. This pre-processing procedure was done offline and only once, guaranteeing that there is no chance of missing a melody existing in any channel of a song that may be similar to melodies of channels of other songs.

5.3.1.2 Evaluation

The classifier that is used for genre classification is the k-NN. First, we created the similarity matrix of all songs to all others (of size 100x100), based on the procedure mentioned in Section 5.2. Then, each of the 100 songs was considered to be a query to which we applied k-NN classification, for k in [1,10]. A query is classified correctly if the majority of its k nearest neighbors belong to the same genre as that of the query, and the measure that we used for evaluating our framework is the *classification accuracy*, which is defined as the percentage of the query songs that were classified correctly. Clearly, we are interested in a large number of queries for which there is clear majority for one genre in their k neighbors and are classified correctly.

In case that for a query song there are more than one genres with the highest number of votes in the returned k-NN results, then there is no clear majority for a genre (independently of whether that genre is the correct one for the query or not). To resolve ties of such cases we applied the following scheme. In the ordered

similarity scores of the k-NN that are returned, the output genre of the classifier is the first genre for which the maximum number of votes is reached (by taking one by one the k neighbors and increasing the counter of the genre they belong to). This is an intuitively good approach for the case of ties, instead of, for example, randomly picking one of the genres that has the maximum number of votes or, even worse, not classifying the query to any genre (and just return all tied genres as possible answers). We believe that further looking at the structure of the songs may provide more accurate results.

With regard to the parameters of SMBGT, the number of elements allowed to be skipped in a query Q was set to $\beta = 6$, and for the target sequence X to which Q is compared was set to $\alpha = 5$. For ϵ_1^f absolute variable tolerance was studied with $t = 0.2$ (Equation 3.4). In addition, the maximum matching range r was set to 1.2 and the minimum number of matching elements δ to 0.1. The reason for instantiating the parameters of SMBGT with these values is because they have been proved to provide the best accuracy for the QBH application (Chapter 3). Although QBH is not the target application in this chapter, we believe that these values should be tested for the genre classification as well. As part of future work, cross-validation can be done to obtain the best combination of parameter values.

Experiments were run on an AMD Opteron 8220 SE processor at 2.8GHz, and implemented in Matlab.

5.3.2 Experimental Results

In Table 5.2 we present the classification accuracies that we obtained for several values of k in the k-NN classification. The "Non-Tied" column shows the number of queries (out of the 100) for which the classifier could clearly decide and return one genre, i.e., there was only one clear winner genre with the maximum number of

Table 5.2: Classification accuracies with k-NN for the dataset of Table 5.1. For each value of k in [1,10] we present the following statistics. The number of songs that were classified having a clear winner genre (column "Non-Tied"), and having more than one tied genres to classify them to (column "Tied"). In addition, for each of the aforementioned columns, the number of songs that were correctly classified is shown in columns "Non-Tied Classified" and "Tied Classified", respectively, along with the corresponding classification accuracies (columns "Accuracy (Non-Tied)" and "Accuracy (Tied)"). The final classification accuracy for each k is also presented in column "Accuracy (Total)".

k	Non-Tied	Non-Tied Classified	Accuracy (Non-Tied) (%)	Tied	Tied Classified	Accuracy (Tied) (%)	Accuracy (Total) (%)
1	100	31	31	0	0	NaN	31
2	30	12	40	70	19	27.14	31
3	72	26	36.11	28	10	35.71	36
4	77	27	35.06	23	10	43.48	37
5	76	29	38.16	24	8	33.33	37
6	66	25	37.88	34	14	41.18	39
7	80	32	40	20	7	35	39
8	83	31	37.35	17	7	41.18	38
9	88	34	38.64	12	5	41.67	39
10	75	30	40	25	10	40	40

votes. Column "Non-Tied Classified" gives the number of queries that were correctly classified out of those for which there was a clear winner genre by the classifier (as indicated by column "Non-Tied"). "Accuracy (Non-Tied)" provides the classification accuracy for the queries that were correctly classified when there was a clear winner genre. In other words, for a specific k it is the division of the number in column "Non-Tied Classified" with that of column "Non-Tied". Column "Tied" is basically the remaining number of query songs up to 100 when we consider column "Non-Tied". "Tied Classified" shows the number of queries that were correctly classified when there were more than one genres with the same maximum number of votes in the returned results (ties) and we treated them following the scheme described in Section 5.3.1.2. In "Accuracy (Tied)" we present the classification accuracy obtained when dividing column "Tied Classified" with "Tied". Finally, the total classification

accuracy is shown in "Accuracy Total", which refers to the total number of queries that were classified correctly (whether they had a clear winner genre in the returned results or after resolving ties).

From Table 5.2 we can observe that the best accuracy for non-tied queries is achieved for $k = 10$ and $k = 7$, and is 40%. We note here that, although 40% accuracy is also achieved for $k = 2$ the total number of non-tied queries is only 30, and thus it is not considered to be a reliable value for accuracy. Regarding the accuracies for "tied" queries the best ones are obtained for $k = 4$ with 43.48% and $k = 9$ with 41.67%. Finally, the best total accuracy is 40% for $k = 10$. According to these accuracies, we observe that the best value for k among all tested values is 10, especially if we observe the total classification accuracy, which is essentially the measure returned to the user by an automatic genre classification system.

There are several reasons for not getting total classification accuracies greater than 40%, while also requiring to look at the %10 of the database songs' scores for each query ($k = 10$) to attain this accuracy. First of all, there is a significant percentage of queries for which we have to resolve ties. This indicates that a more thorough study of the structure of songs should be done so as to apply a more clever mechanism to deal with ties, or even use another similarity measure for such cases that will be based on low-level features of the songs. What is more, we perform genre classification with k-NN, which is a simple classifier that does not require any kind of training. However, since in the music domain describing a genre requires knowledge of not only intrinsic properties of songs but also cultural features (that greatly influence the characterization of songs into genres), more complex and trained classifiers on larger databases of songs should provide more promising classification accuracies.

Another important observation that we came up with is that for the majority of k values that we experimented with the best "Accuracy (Non-Tied)" per genre was

achieved in the following order: Blues, Rock, Pop, and Classical. This is indeed the case in music. Blues songs have a very well-defined compositional structure, followed by Rock songs, which can also be characterized to have a particular style, but less well-defined than Blues. Additionally, Pop songs are much harder to be treated as having one style, since they cover a wide variety of "popular" music. Finally, Classical music is the hardest genre to classify songs to, because the structure and especially the melody of each piece depends basically on the personal taste and the music era of the composer. According to these music observations that are confirmed by our experiments, SMBGT is proved to be very promising with regard to discriminating genres based on their structure.

5.4 Conclusions and Future Work

We have applied SMBGT to perform music genre classification with the k-NN classifier. Music genre classification can be highly applicable to assistive environments since music can be seen as a means of stimulating and focusing attention for people with disabilities. Since both the definition of genres and the discrimination among them is in general very vague and problematic, the classification accuracies that we got for a set of 100 queries were not very high for different values of k. Thus, to improve the classification accuracy there are several aspects that could be considered for future work.

First, instead of using a simple classifier that does not require training, diverse and independent trained classifiers (and combination of them) can be used. Secondly, features can be extracted from short segments of music pieces to create multidimensional sequences, and then evaluate the performance of classifier(s) using SMBGT. Additionally, different parameters for SMBGT could be tested, for example by using

cross-validation. What is more, polyphonic music could be analyzed in order to see if the compositional structure of songs is meaningful for the classification task. Finally, to build a more realistic automatic music genre classification system bigger datasets should be collected covering more genres (and even splitting them to subgenres).

CHAPTER 6

QUERY-SENSITIVE DISTANCE MEASURE SELECTION FOR TIME SERIES NEAREST NEIGHBOR CLASSIFICATION

Time series data have become ubiquitous during the last decades. Sequences of numerical measurements are daily produced at regular or irregular time intervals in vast amounts in almost every application domain, such as stock markets, medicine, sensor networks (cameras, accelerometers, implantable sensors measuring vital statistics of a patient, RFID's, and devices measuring temperature and humidity), moving objects (e.g., in traffic and astronomy), scientific experiments, and biology. As a result, there has been a significant amount of research on mining and querying time series. However, due to the importance of querying time series, there is still a need for new techniques for classification, clustering, indexing, and approximation of time series [123].

Large databases of time series can be exploited so as to extract knowledge on what has happened in the past or to recognize what is happening in the present. As an example, consider that we have a sequence of measurements and we would like to identify the sequences of the database that are the most similar to that sequence. This would help in determining, e.g., the state of the environment or of a patient, the traffic congestion, financial situation of countries, or even the genre of a music piece [122]. In addition, given examples or models of specific important events in the present, we may be interested in identifying similar events that have happened in the past.

Specifically, a fundamental task in knowledge discovery is 1-Nearest Neighbor (NN) classification. According to this task, given a dataset of time series belonging to certain *categories/classes* (also known as *training* time series) and an unclassified query time series, we identify its class by looking at the class of its nearest neighbor. The nearest neighbor is found by computing the distance between the query and all time series in the dataset via a distance measure, and then selecting the time series with the smallest distance.

In such a setting, the key challenge is to define or choose an appropriate distance measure to perform whole sequence matching between each time series in the collection and the query, which is expected to retrieve the correct class label for a given query. It can be easily understood that the selection of the distance measure to be used for comparing time series is critical, as it essentially decides whether a time series is a good match for the query or not, influencing the classification accuracy (percentage of time series correctly classified).

As mentioned in Section 2.4, several distance and similarity measures have been developed for performing whole sequence matching between two time series, such as DTW [33], EDR [39], ERP [40], TWED [44], and MSM [42]. Moreover, alternative methods have been developed for the same problem, that focus on global or local structural similarity, such as SpaDe [28], Shapelets [23], DFT [29], Bag-Of-Patterns [31], and SAX [30]. Nonetheless, none of these methods is guaranteed to be optimal for the task of NN classification. In other words, some measure or method may fail to correctly classify some queries while some other may be successful with these queries, and vice versa.

We will now illustrate the previous observation with a motivating example. Suppose we have a collection of 45 time series datasets from a variety of application domains and a *pool* of distance measures. For simplicity, we consider only two mea-

sures in our pool, e.g., DTW and MSM. More information about these datasets can be found in Section 6.4.1. Our main question is the following: given an unlabeled query, which of the two measures is more likely to classify it correctly using our data collection? Unfortunately the answer to this question is not straightforward.

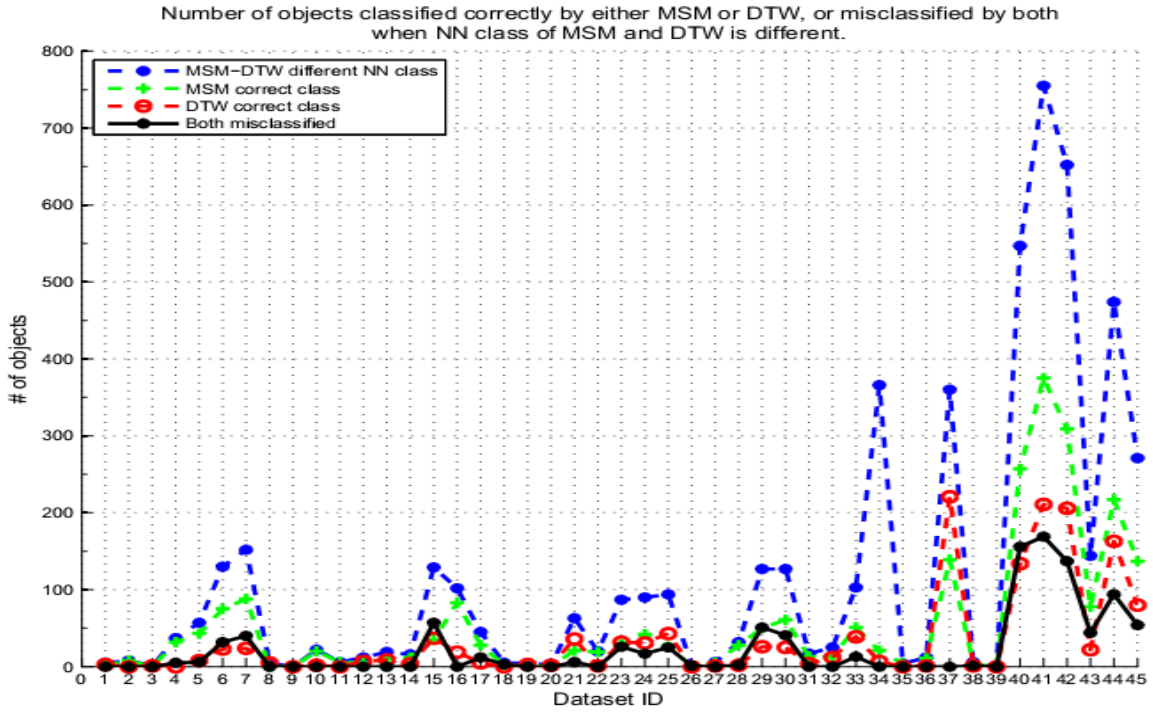


Figure 6.1: An example that illustrates the challenging nature of our problem. We used 45 datasets from the UCR time series repository (x axis). On the y axis we show the number of time series for each dataset that are classified correctly by either DTW or MSM, or misclassified by both, when the class of the NN object is different between DTW and MSM. It can be seen that the number of time series correctly classified by DTW is comparable to that of MSM.

In Figure 6.1 we present for each of the 45 datasets in our collection the number of time series where the class of the NN for DTW is different than that of MSM (blue dotted line), and out of these time series how many are classified correctly by either DTW or MSM (green and red dotted lines, respectively), or misclassified by both of

them (black line). Thus, the black line essentially represents the number of time series for which there is no room for improvement, or, in other words, whatever measure (DTW or MSM) is selected they cannot be classified correctly. We observe that there is no clear winner between the two measures. In other words, for some datasets DTW performs better than MSM, while for some other datasets MSM is better. Ideally, we would like to select the measure that would correctly classify a given query time series, if that is possible by any of the two measures.

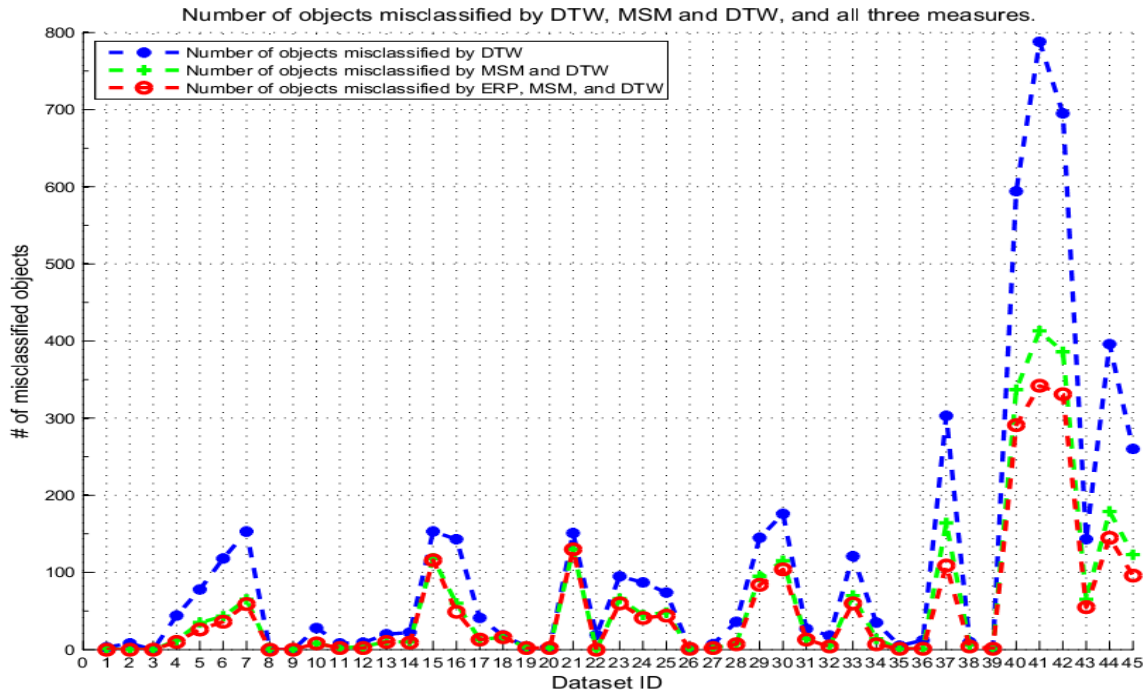


Figure 6.2: An example that illustrates the number of available distance measures may affect the classification accuracy. We used 45 datasets from the UCR time series repository (x axis). On the y axis we show the number of time series for each dataset that are misclassified by all measures in the pool. It can be seen that as the number of measures in the pool increases (from 1 to 3), the number of time series that are still incorrectly classified by all measures in the pool decreases for most of the datasets, or remains the same.

Furthermore, we note that the more distance measures we have available to choose from, the higher the chance of correctly classifying the given query. Figure 6.2 illustrates results of our study when more distance measures are added to our pool. Specifically, we show for each dataset the number of misclassified time series when the pool consists of either only DTW (blue asterisk), or MSM and DTW (black cross), or ERP, MSM, and DTW (red circle). We can argue that the more distance measures we add to the pool, the higher the classification accuracy may be if we manage to select the proper measure. For example, for datasets with IDs 41 and 44 there are 788 and 396 time series, respectively, misclassified when our pool consists of only DTW. When MSM is also added to the pool, the numbers of time series misclassified by both MSM and DTW drop to 413 and 179, respectively. Finally, when ERP is also included in the pool, the numbers of time series that are misclassified by all three measures drop further to 342 and 145, respectively, for the same datasets.

We study the following challenging problem: given a pool of distance measures and a query time series, identify the most appropriate distance measure for NN classification for the given query. Note that our focus is not to find the best distance measure or "golden" standard method for time series NN classification, but the most suitable one for a given query in a predefined pool of measures.

The contributions of our work can be summarized as follows:

- To the best of our knowledge, there exists no relevant work towards selecting the best measure, given a query and a pool of measures, for time series NN classification.
- We propose a framework for solving the aforementioned problem, where the selection of the appropriate distance measure is performed via statistical significance testing on the pool of measures using a set of training time series.

The framework is also query-sensitive, i.e., the selected distance measure can be different for each query.

- Within this framework we propose two methods that employ different schemes in order to select the appropriate training time series for the statistical test. The intuition of the first method is that, if a query is much closer to its nearest neighbor, which has a certain class, than to the closest object of a different class, then we are more confident that the query belongs to the class of the nearest neighbor. The idea behind the second method is to count the number of objects that belong to the same class as that of the nearest neighbor, or, in other words, to measure the homogeneity of an object's neighborhood.
- We provide an extensive experimental evaluation of the proposed methods against a baseline approach on a large collection of 45 time series datasets from the UCR repository [124], which shows that our framework can achieve at least as good or better classification accuracies than the baseline on up to 35 datasets. This suggests that our framework can be highly competitive in terms of NN classification accuracy by using a pool of measures instead of using only one of them.

6.1 Related Work

All the whole sequence matching measures presented in Section 2.4 could be used in our pool of measures for NN classification. We also have to note that, although the time series representation methods presented in Section 2.3 are not directly applicable to our problem setting since we are particularly interested in distance measures, they could be employed for NN classification in time series databases, and hence could as well be considered in our pool of measures.

In order to deal with the curse of dimensionality when performing NN classification, a linear discriminant analysis has been proposed [125] to estimate an effective metric for computing neighborhoods. Based on centroid information, local decision boundaries are determined, the neighborhoods are shrunk in directions orthogonal to these boundaries, and any NN classifier is performed on the modified neighborhoods. Similarly to this approach, Domeniconi et al. [126] propose an adaptive NN classification method to minimize estimation bias in high dimensions. Based on Chi-squared distance analysis, a flexible metric, which depends on query locations in the feature space, is estimated for computing neighborhoods that are constricted along the most influential feature dimensions. Both of these methods are designed for patterns that are represented as vectors in Euclidean space, and thus they are not applicable to time series, which is the focus of this work. Furthermore, they do not select the most appropriate query-based measure from a pool of measures, rather they try to find the most influential dimensions. Finally, Athitsos et al. [127] proposed a method for approximate nearest neighbor retrieval by mapping objects from the original space to a real vector space using a set of reference objects. The query-sensitive nature of that work is that the distance measure used in the vector space is a weighted Lp norm, where the weights are learned via boosting during an expensive pre-processing step. In our work, we tackle a much different problem, that of classifying a time series by selecting the most appropriate distance measure for the given query out of a pool of measures.

6.2 Background

Given a distance measure $dist$, the distance between two time series X and Y is defined as a function $d_{\text{dist}}(X, Y)$. Additionally, $\mathcal{L} = \{dist_1, \ldots, dist_n\}$ defines a *pool of distance measures*, where each $dist_x$, $x = 1, \ldots, n$, is a distance measure.

Problem Setting. Given a collection of time series $\mathcal{T} = \{X_1, \ldots, X_N\}$, a pool of distance measures $\mathcal{L}$, and a query time series Q, we want to identify the distance measure $dist_x \in \mathcal{L}$ that is most suitable to perform NN classification for Q.

We explore three time series distance measures, which are used to construct our pool $\mathcal{L}$. These measures are: Dynamic Time Warping (DTW) [33], Edit distance with Real Penalty (ERP) [40], and Move-Split-Merge (MSM) [42]. The selection of these measures is based on the following three rationales: (1) DTW is one of the most commonly used and studied distance measures for time series matching, (2) ERP is a variant of DTW and Edit Distance that fixes the non-metric property of DTW, and (3) MSM is a metric distance measure that has been proposed very recently and is shown to outperform ERP and DTW in terms of NN classification accuracy for several datasets. These three measures are described next in more detail.

It should be noted that we do not intend to claim that these are the best measures among all existing ones. Our aim is to demonstrate that the proposed framework can achieve competitive performance to existing measures in terms of NN classification accuracy, since instead of using only a single distance measure it exploits the strengths of each measure in the pool and identifies the most appropriate one for a given query. Hence, other measures could be used alternatively without any change in the framework.

For the convenience of the reader, in Table 6.1 we present the notation used throughout this chapter.

Table 6.1: Notation Table.

Notation	Explanation
$X = (x_1, \ldots, x_{\|X\|})$	A 1-dimensional time series of length $\|X\|$.
$dist$	A distance measure.
$d_{\text{dist}}(X, Y)$	A distance function computing $dist$ between X and Y.
$\mathcal{L} = \{dist_1, \ldots, dist_n\}$	A pool of distance measures.
$\mathcal{T} = \{X_1, \ldots, X_N\}$	A collection of N training time series.
$\mathcal{D}_{X_i,\mathcal{T}}^{dist_x}$	Set of distances between X_i and all time series in $\mathcal{T}$ using $dist_x$.
f^{scheme}	A scheme function.
s_i	The score given by a scheme function f^{scheme} for X_i.
o_i	The closest to X_i training time series for some $dist_x$.
cr_i	The 1-NN classification result for X_i using $dist_x$.
s	An array of all s_i values for some $dist_x$.
o	An array of all o_i values for some $dist_x$.
cr	An array of all cr_i values for some $dist_x$.
s'^{dist_x}	The sorted values of s for some $dist_x$.
o'^{dist_x}	The sorted indices of o based on the sorting of s'^{dist_x}.
cr'^{dist_x}	The sorted indices of cr based on the sorting of s'^{dist_x}.
Q	A query time series.
Q_o	The closest to Q training time series.
Q_s	The score given by a scheme function f^{scheme} for Q.
$pos(Q_s, s'^{dist_x})$	The position of Q_s in s'^{dist_x}.
T	The T-neighborhood parameter.
v_{dist_x}	The T-neighborhood classification vector for some $dist_x$.
$P_E^{dist_x}$	The 1-NN classification error probability for some $dist_x$.
$dist_{min}$	The measure with the lowest classification error probability.
$\mathcal{V}^T$	The set of T-neighborhood classification vectors for all measures.
$pval(\mathcal{V}^T, \alpha)$	The p-value computed by ANOVA with significance threshold α.

6.2.1 Dynamic Time Warping

DTW identifies an optimal alignment between two time series and computes the matching cost of that alignment in quadratic computational time. Each time series element is allowed to match with at least one element of the other time series, allowing for local stretching and shrinking along the time axis. Given two time series

X and Y, their DTW distance $d_{\text{DTW}}(X, Y)$ is defined recursively using a dynamic programming matrix [1] *Cost* of size $(|X| + 1) \times (|Y| + 1)$. A *null* element is added at the beginning of X and Y, and it matches the other null element with zero score and any other element with a score of ∞. Let $Cost_{i,j}$ denote the element at the i-th row and j-th column of *Cost*. Denoting with $L_p(x_i, y_j)$ the L_p norm based distance measure of x_i and y_j, we define $d_{\text{DTW}}(X, Y)$ as follows:

Initialization:

$$Cost_{0,0}(X, Y) = 0 \ , \ Cost_{0,j}(X, Y) = \infty \ , \ Cost_{i,0}(X, Y) = \infty \ .$$

Main Loop:

$$Cost_{i,j}(X, Y) = L_p(x_i, y_j) + \min \left\{ \begin{array}{l} Cost_{i,j-1}(X, Y), \\ Cost_{i-1,j}(X, Y), \\ Cost_{i-1,j-1}(X, Y) \end{array} \right\}$$

$$\forall (i = 1, \ldots, |X|; j = 1, \ldots, |Y|) \ .$$

Output:

$$d_{\text{DTW}}(X, Y) = Cost_{|X|,|Y|}(X, Y) \ .$$

6.2.2 Edit distance with Real Penalty

ERP [40] is a metric distance measure with quadratic time complexity that can be seen as a variant of the L_1 norm, EDR, and DTW. The main advantage over DTW is that it satisfies the triangle inequality. Specifically, while DTW replicates the value of the previous element when a gap is introduced in either time series, ERP applies a nonnegative constant penalty g for computing the distance between the gap and the corresponding element from the other time series. In the case of non-gap elements, the matching penalty is simply their L_1 norm. Given a nonnegative constant parameter

g, function G is used in computing values for the $Cost$ array, where $G(x_i, y_j)$ is defined as follows:

$$G(x_i, y_j) = \begin{cases} |x_i - y_j|, \text{ if } x_i \text{ and } y_j \text{ are not gaps,} \\ |x_i - g|, \text{ if } y_j \text{ is a gap,} \\ |g - y_j|, \text{ if } x_i \text{ is a gap .} \end{cases}$$

The distance $d_{\text{ERP}}(X, Y)$ is now defined as follows:

Initialization:

$$Cost_{0,j}(X, Y) = \sum_{j=1}^{|Y|} |y_j - g|$$

$$Cost_{i,0}(X, Y) = \sum_{i=1}^{|X|} |x_i - g|$$

Main Loop:

$$Cost_{i,j}(X, Y) = \min \left\{ \begin{array}{l} Cost_{i-1,j-1}(X, Y) + G(x_i, y_j), \\ Cost_{i-1,j}(X, Y) + G(x_i, gap), \\ Cost_{i,j-1}(X, Y) + G(gap, y_j) \end{array} \right\}$$

$$\forall (i = 1, \ldots, |X|; j = 1, \ldots, |Y|) \ .$$

Output:

$$d_{\text{ERP}}(X, Y) = Cost_{|X|,|Y|}(X, Y) \ .$$

6.2.3 Move-Split-Merge

MSM [42] is a metric time series distance measure, robust to misalignments, translation invariant, and has again a quadratic computational time complexity. Given two times series X and Y, MSM transforms X to Y by employing three fundamental operations: Move, Split, and Merge. The Move operation changes the value of a single point of the time series, the Split operation splits a single point of the

time series into two consecutive points that have the same value as the original point, while the Merge operation merges two successive equal values into one. Thus, the MSM distance between X and Y, $d_{\text{MSM}}(X, Y)$, is defined as the cost of the lowest-cost transformation of X to Y. Similarly to DTW and ERP, given two time series X and Y, their MSM distance can be computed using dynamic programming. For each (i, j) such that $1 \leq i \leq |X|$ and $1 \leq j \leq |Y|$, $Cost_{i,j}$ is defined to be the MSM distance between the first i elements of X and the first j elements of Y. Given a nonnegative constant parameter c, function C is used in computing values for the $Cost$ array, where $C(x_i, x_{i-1}, y_j)$ is:

$$C(x_i, x_{i-1}, y_j) = \begin{cases} c, \text{ if } x_{i-1} \leq x_i \leq y_j \text{ or } x_{i-1} \geq x_i \geq y_j \\ c + \min(|x_i - x_{i-1}|, |x_i - y_j|), \text{ otherwise} \end{cases}$$

The distance $d_{\text{MSM}}(X, Y)$ is now defined as follows:

Initialization:

$$Cost_{1,1}(X, Y) = |x_1 - y_1| \ .$$

$$Cost_{i,1}(X, Y) = Cost_{i-1,1}(X, Y) + C(x_i, x_{i-1}, y_1) \ .$$

$$Cost_{1,j}(X, Y) = Cost_{1,j-1}(X, Y) + C(y_j, x_1, y_{j-1}) \ .$$

$$\forall (i = 2, \ldots, |X|; j = 2, \ldots, |Y|) \ .$$

Main Loop:

$$Cost_{i,j}(X, Y) = \min \left\{ \begin{array}{l} Cost_{i-1,j-1}(X, Y) + |x_i - y_j|, \\ Cost_{i-1,j}(X, Y) + C(x_i, x_{i-1}, y_j), \\ Cost_{i,j-1}(X, Y) + C(y_j, x_i, y_{j-1}) \end{array} \right\}$$

$$\forall (i = 2, \ldots, |X|; j = 2, \ldots, |Y|) \ .$$

Output:

$$d_{\text{MSM}}(X, Y) = Cost_{|X|,|Y|}(X, Y) \ .$$

6.3 Query-sensitive Measure Selection

In this section, we present a framework for solving the problem defined in Section 6.2, and also two methods that are based on this framework.

6.3.1 Measure-selection Framework

The proposed framework consists of two steps: the *offline* and the *online* step.

6.3.1.1 Offline step

This is a pre-processing step and is performed only once. Given the set $\mathcal{T}$ of N training time series (also referred to as training objects) and a distance measure $dist_x$, we first compute the distance of each time series $X_i \in \mathcal{T}$ to all other time series $X_j \in \mathcal{T}$, resulting in the following set of distances:

$$\mathcal{D}^{dist_x}_{X_i,\mathcal{T}} = \{d_{dist_x}(X_i, X_j) | \forall X_j \in \mathcal{T}, i \neq j\} \ . \tag{6.1}$$

Next, for each $X_i \in \mathcal{T}$ we determine its closest time series o_i based on $dist_x$, i.e.,

$$o_i = \text{argmin}\left\{\mathcal{D}^{dist_x}_{X_i,\mathcal{T}}\right\}. \tag{6.2}$$

Then, the set of distances $\mathcal{D}^{dist_x}_{X_i,\mathcal{T}}$ is passed into a *scheme* function $f^{scheme}(\cdot)$. For the remainder of this subsection we will use the term scheme function as a black box that, given a time series X_i and the pairwise distances of X_i to all other objects in the training set $\mathcal{T}$, produces a score s_i based on these distances. Note that in our current

case where X_i is already part of the training set, we only consider the remaining $N-1$ objects in $\mathcal{T}$. The score s_i is given by:

$$s_i = f^{scheme}(X_i, \mathcal{D}^{dist_x}_{X_i,\mathcal{T}}) \ . \tag{6.3}$$

Further details about schemes and their functions are discussed in Section 6.3.2.

Finally, we treat each X_i as a query and determine whether it was correctly classified by $dist_x$ using the 1-NN classifier, that means comparing its class with that of o_i. The result of this comparison is stored in cr_i:

$$cr_i = \begin{cases} 0, \text{ if } X_i \text{ is correctly classified,} \\ 1, \text{ otherwise.} \end{cases} \tag{6.4}$$

As a result, for each $X_i \in \mathcal{T}$ we store three values: o_i, s_i, and cr_i. Effectively, this produces three arrays of size N:

- $o = [o_1 \dots o_N]$: for each X_i the index of its closest training time series o_i,
- $s = [s_1 \dots s_N]$: for each X_i the score s_i given by the scheme function f^{scheme}, and
- $cr = [cr_1 \dots cr_N]$: for each X_i the classification result cr_i determined by the 1-NN classifier using $dist_x$.

In addition, for ease of the online step, array s is sorted in ascending order, while the indices of arrays o and cr are rearranged accordingly. This sorting procedure results in the rearranged arrays $o' = [o'_1 \dots o'_N], s' = [s'_1 \dots s'_N]$, and $cr' = [cr'_1 \dots cr'_N]$.

Finally, the offline step is performed for each distance measure $dist_x \in \mathcal{L}$, and the corresponding arrays are denoted as o'^{dist_x}, s'^{dist_x}, and cr'^{dist_x}, respectively.

6.3.1.2 Online step

At runtime, given a query time series Q, we identify the most appropriate distance measure to classify Q via statistical significance testing using the set of training objects $\mathcal{T}$.

Distance computation. First, for each distance measure $dist_x \in \mathcal{L}$, the distances of Q to all training objects $X_j \in \mathcal{T}$ are computed, resulting in the following set:

$$\mathcal{D}_{Q,\mathcal{T}}^{dist_x} = \{d_{dist_x}(Q, X_j) | \forall X_j \in \mathcal{T}\} \ . \tag{6.5}$$

Similarly to the offline step, the closest training time series is identified, i.e.,

$$Q_o = \text{argmin}\left\{\mathcal{D}_{Q,\mathcal{T}}^{dist_x}\right\}, \tag{6.6}$$

and the score of the scheme function is recorded, i.e.,

$$Q_s = f^{scheme}(Q, \mathcal{D}_{Q,\mathcal{T}}^{dist_x}) \ . \tag{6.7}$$

The challenge now is to determine the class of Q by deciding which distance measure to "trust" for our NN classifier.

Query classification. Next, we identify the position of Q_s in each s'^{dist_x} (computed in the offline step) as follows:

$$pos(Q_s, s'^{dist_x}) = p, \text{ if } \begin{cases} s_p'^{dist_x} = Q_s \text{ and } 1 \leq p \leq N, \\ \text{or } s_p'^{dist_x} < Q_s < s_{p+1}'^{dist_x} \text{ and } 1 \leq p < N, \\ \text{or } s_N'^{dist_x} < Q_s, \\ \text{or } s_1'^{dist_x} > Q_s \ . \end{cases} \tag{6.8}$$

In other words, we identify the position p in s'^{dist_x} with value equal to Q_s. If Q_s does not appear exactly in s'^{dist_x}, then p corresponds to the position of the last value that is smaller than Q_s (except for the last case).

The rationale here is that we expect that objects given similar scores by the scheme function should have similar classification results for a given distance measure. Hence, we should focus on a "neighborhood" of Q_s in s'^{dist_x}.

In particular, for all distance measures in $\mathcal{L}$ we define a parameter T that specifies the *T-neighborhood* of Q. We select $T-1$ training time series from the left and T from the right side of $pos(Q_s, s'^{dist_x})$ in s'^{dist_x}, including $pos(Q_s, s'^{dist_x})$ itself. This results in $2 * T$ time series in total (for each $dist_x \in \mathcal{L}$). In case that there are are less than $T-1$ or T training time series on the left or right side of $pos(Q_s, s'^{dist_x})$, respectively, we fill out the remaining time series from its other side, so that there are always $2 * T$ time series. The classification result for each of the $2 * T$ time series can be directly retrieved from cr'^{dist_x}.

Consequently, for each distance measure $dist_x \in \mathcal{L}$ we can now extract the following vector of classification result values:

$$v_{dist_x} = (cr'^{dist_x}_{pos(Q_s, s'^{dist_x})-T+1}, \ldots, cr'^{dist_x}_{pos(Q_s, s'^{dist_x})+T}) \tag{6.9}$$

Using vectors v_{dist_x} for all $dist_x \in \mathcal{L}$, we compute the classification error probability for each distance measure as follows:

$$P_E^{dist_x} = \frac{\|v_{dist_x}\|_1}{2 * T} \tag{6.10}$$

Next, we select the distance measure $dist_{min}$ with the lowest such probability and use that measure for the NN classification task for the given query:

$$dist_{min} = \underset{dist_x \in \mathcal{L}}{\operatorname{argmin}} \{P_E^{dist_x}\}. \tag{6.11}$$

Finally, Q is assigned the class of the closest training time series, as defined by $dist_{min}$.

The intuition for creating vectors v_{dist_x} and computing the error probabilities $P_E^{dist_x}$ for each measure $dist_x \in \mathcal{L}$ for a given T is the following: given two measures $dist_i$ and $dist_j$, if there are more misclassified training time series in the T-neighborhood of Q using $dist_i$ than in the T-neighborhood of Q using $dist_j$, then this leads in a higher value of $P_E^{dist_i}$. More precisely, the T-neighborhood for $dist_i$ does not provide a good guarantee for correct classification of Q, in contrast to $dist_j$ that classifies more training objects around Q correctly according to the selected scheme. Consequently, $dist_j$ is more suitable for Q, since it is less likely to produce wrong classification.

Defining the T-neighborhood. An appropriate value for T is chosen for all distance measures $dist_x \in \mathcal{L}$ based on statistical significance testing. Specifically, we use the ANalysis Of VAriance (ANOVA) test [128], which is a generalization of the t-test when more than two groups are analyzed.

First, for each $T \in [1, \lceil N/2 \rceil]$, we construct the corresponding set of vectors $\mathcal{V}^T = \{v_{dist_1}, \ldots, v_{dist_n}\}$ for all distance measures in $\mathcal{L}$. Note that the vectors are constructed around $pos(Q_s, s'^{dist_x})$ as described in the previous section. Next, we perform ANOVA with statistical significance threshold α, which produces a p-value:

$$pval(\mathcal{V}^T, \alpha) = ANOVA(\mathcal{V}^T, \alpha) \ , \forall T \in [1, \lceil N/2 \rceil] \ . \tag{6.12}$$

Using Equation 6.12, we assign our final threshold T with the value that corresponds to the smallest p-value:

$$T = \underset{T \in [1, \lceil N/2 \rceil]}{\operatorname{argmin}} \left\{ pval(\mathcal{V}^T, \alpha) \right\} . \tag{6.13}$$

The rationale here is that this way of determining the T-neighborhood gives the most statistically significant NN classification accuracy results when using the set of measures in $\mathcal{L}$ and the training time series for all possible neighborhoods of Q. T is

then used for the computation of the classification error probabilities (Equation 6.10), and will hence determine the most appropriate measure (Equation 6.11) for the given query Q.

It has to be mentioned that if more than one T values provide the same lowest probability the smallest one is chosen to define the query neighborhood. Also, note that, by construction, in each v_{dist_x} vector there are always $2 * T$ objects.

In Figure 6.3 we illustrate the main steps of the proposed framework.

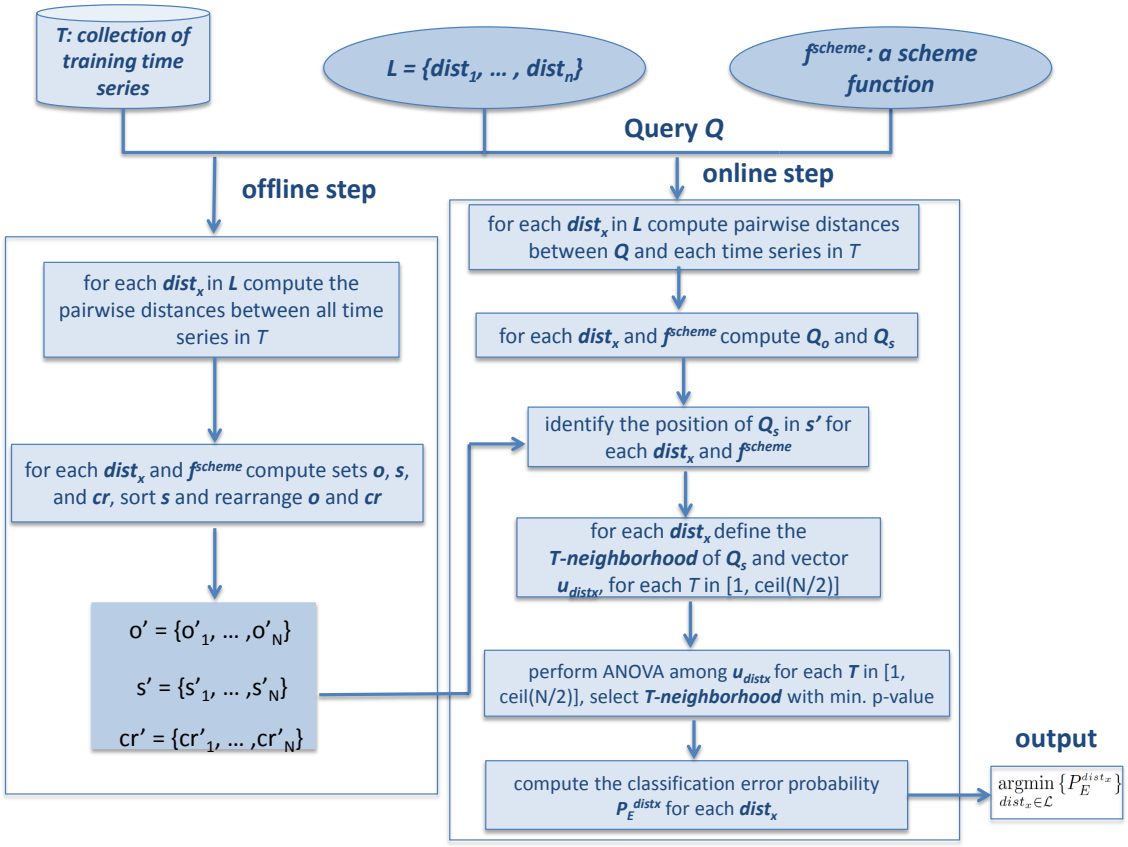


Figure 6.3: Illustration of the offline and online steps of the proposed query-sensitive measure selection framework.

6.3.2 Methods

We present two methods that can be used within our framework for constructing the neighborhood of training objects for a given query. Each method is characterized by a *scheme*, which is a technique for computing the values in the array s and the value of Q_s used in the framework. Each scheme uses a function f^{scheme} to compute these values. Note that we first describe a *basic* scheme that is used as a building block by the two proposed schemes.

6.3.2.1 Scheme 0: Basic

This basic scheme constructs s and Q_s based exclusively on *distances*. The intuition here is the following: if a query Q is very close to its NN, then we can be more confident that Q indeed belongs to the class of its NN. If we find that Q is very close to its NN for one distance measure $dist_x$ and not for the others, then we have a reason to trust $dist_x$ more than the other measures, for the specific query. The neighborhood of a query is thus created by the training objects for which the distance from their closest object is close to the distance of the query to its closest training object.

More specifically, in the offline step, after the computation of the set of distances $\mathcal{D}_{X_i,\mathcal{T}}^{dist_x}$ of the training objects $X_i \in \mathcal{T}$ for all distance measures $dist_x \in \mathcal{L}$, we apply the scheme function f^0 as follows:

$$s_i = f^0(X_i, \mathcal{D}_{X_i,\mathcal{T}}^{dist_x}) = min(\mathcal{D}_{X_i,\mathcal{T}}^{dist_x}) \ . \tag{6.14}$$

In other words, the value s_i returned by the scheme function is the distance of X_i to its closest time series o_i. In a similar manner, during the online step, Q_s is computed for each $dist_x$ using the scheme function as follows:

$$Q_s = f^0(Q, \mathcal{D}^{dist_x}_{Q,\mathcal{T}}) = min(\mathcal{D}^{dist_x}_{Q,\mathcal{T}}) \ . \tag{6.15}$$

As it can be seen from the above equation, Q_s corresponds to the distance of Q to its closest training object. The remaining steps of the online step proceed as described in the previous section.

6.3.2.2 Scheme I: Distance ratio-based

The first technique for constructing s is an extension of the basic scheme. It is computed by using the ratio of distance from the closest object to the distance from the next closest object with a different class. The intuition behind this scheme is this: if a query Q is much closer to its nearest neighbor, which has a certain class, than to the closest object of a different class, then we are more confident that Q indeed belongs to the class of the nearest neighbor. If this happens for one distance measure $dist_x$ and not for the others, then we have a reason to trust $dist_x$ more than the other measures, for the specific query Q.

In particular, for each X_i the scheme identifies its closest object o_i by Equation 6.2. Furthermore, $\mathcal{D}^{dist_x}_{X_i,\mathcal{T}}$ is sorted, resulting in $\mathcal{D}'^{dist_x}_{X_i,\mathcal{T}}$, and the corresponding objects are scanned until an object of a different class than that of o_i is found; let us call this object X_l and let l be the index of this object in $\mathcal{D}'^{dist_x}_{X_i,\mathcal{T}}$.

The value produced by the corresponding scheme function f^I is given below:

$$\begin{aligned} s_i = f^I(X_i, \mathcal{D}^{dist_x}_{X_i,\mathcal{T}}) &= f^0(X_i, \mathcal{D}^{dist_x}_{X_i,\mathcal{T}}) / d_{dist_x}(X_i, X_l) \\ &= d_{dist_x}(X_i, o_i) / d_{dist_x}(X_i, X_l) \ . \end{aligned} \tag{6.16}$$

Finding this ratio for all training objects and then sorting these ratios in ascending order ends up in array s'^{dist_x}.

During the online step, $\mathcal{D}_{Q,\mathcal{T}}^{dist_x}$ is sorted resulting in $\mathcal{D}'^{dist_x}_{Q,\mathcal{T}}$. Then, the value of Q_s is computed as the ratio of its distance from the closest training object Q_o to its distance from the first object that has a different class than that of Q_o, say $Q_{l'}$ with l' being the index of $Q_{l'}$ in $\mathcal{D}'^{dist_x}_{Q,\mathcal{T}}$:

$$\begin{aligned} Q_s = f^I(Q, \mathcal{D}_{Q,\mathcal{T}}^{dist_x}) &= f^0(Q, \mathcal{D}_{Q,\mathcal{T}}^{dist_x}) / d_{dist_x}(Q, Q_{l'}) \\ &= d_{dist_x}(Q, Q_o) / d_{dist_x}(Q, Q_{l'}) \ . \end{aligned} \tag{6.17}$$

Based on the intuition for this method, ideally, if Q_s is small then its T-neighborhood will comprise objects with low ratios, and these objects (or most of them) are not misclassified. Consequently, in such case there is higher probability of classifying the query correctly.

6.3.2.3 Scheme II: Homogeneity-based

Using the same notation as in the previous method, another approach is to count the number of objects from o_i to X_l and from Q_o to $Q_{l'}$, respectively. We note that objects o_i and Q_o are included in the counting, while objects X_l and $Q_{l'}$ are not. Hence, the scheme function f^{II} for a training object X_i is defined as follows:

$$s_i = f^{II}(X_i, \mathcal{D}_{X_i,\mathcal{T}}^{dist_x}) = l - 1 \ . \tag{6.18}$$

And similarly, for the query Q:

$$Q_s = f^{II}(Q, \mathcal{D}_{Q,\mathcal{T}}^{dist_x}) = l' - 1 \ . \tag{6.19}$$

So, essentially, we count the number of objects that belong to the same class as that of the NN. By doing so, we are in practice measuring the *homogeneity* of an

object's neighborhood. Each s'^{dist_x} thus consists of the sorted (in ascending order) homogeneity values of all training objects under measure $dist_x$.

An ideal situation would be to obtain a high value for Q_s making it lie within several objects with high homogeneity values in s'^{dist_x}. This would imply that all of these objects have many neighbors of the same class, and if this class is the correct one for all objects (or for most of them) then the probability of the query being classified correctly would increase.

Since our framework follows a statistical analysis based on the significance produced by ANOVA, there may be scenarios where, although Q_s may be small for the Distance ratio-based or high for the Homogeneity-based scheme in the ideal situations described above, the T-neighborhood with the highest statistical significance may be large. This means that the neighborhood of the query may even include objects that are misclassified. This fact shows that there is a tradeoff between getting a "good" value for the selected scheme and a "proper" relatively small value for T.

It has to be noted that all of the aforementioned scheme-based measure selection methods are based on the following concept: objects of the same class create regions well separated from objects of other classes. This inter-class dissimilarity, however, is not always the case, since there may be outlier objects making the classification task harder whatever the classifier may be.

6.4 Experiments

6.4.1 Experimental Setup

6.4.1.1 Datasets

We experimented on the 45 time series datasets available on the UCR time series archive [124]. The number of training and test time series ("train size", "test

size"), the length of each time series ("seq. length"), and the number of classes ("class num.") for each dataset are shown in Table 6.2.

6.4.1.2 Methods

We evaluated the two proposed methods, i.e., `Distance ratio-based` and `Homogeneity-based` against a baseline competitor method, which is described below and we shall call `Cross Validation`. For completeness, we also show the performance of the `Basic` method. The pool of measures $\mathcal{L}$ included DTW, ERP, and MSM (described in Sections 6.2.1, 6.2.2, and 6.2.3).

We designed the `Cross Validation` method as follows: (1) for each dataset we first computed the classification accuracy for DTW, ERP, and MSM on the training set using leave-one-out cross validation, and (2) the method outputs the classification accuracy on the test set of the measure with the best accuracy on the training set. If more than one measures provide the same highest (best) classification accuracy on the training set, then the accuracy of `Cross Validation` is the average of the accuracies of these measures on the test set.

Regarding the parameters of the measures, MSM has one free parameter, namely c, which is the cost of every Split and Merge operation. For each of the datasets, the value for c was chosen from the set $\{0.01, 0.1, 1\}$, using leave-one-out cross-validation on the training set and comparing the three classification accuracies. In addition, it has been shown that no greater value of c may achieve good classification results, while these values are sufficient to produce very competitive classification accuracies compared to DTW and ERP [42]. For DTW the L_p used was the Euclidean distance, and the penalty g of ERP was set to 0 [40, 42]. Finally, for the statistical significance testing, the significance threshold α of ANOVA was set to 0.05.

6.4.1.3 Evaluation Measures

Since the task at hand is NN classification, for each dataset, we evaluate all of the aforementioned methods in terms of *classification error rate*, which is defined as the percentage of the test time series that are misclassified using the NN classifier.

At this point we note that if in our framework there are ties among the measures in the minimum error probability (computed by Equation 6.10) for a query, we select the measure with the smallest classification error rate on the training set. In case of further tie we use all measures involved in the tie, by taking the average classification result of these measures (recall that 0 corresponds to correct classification, whereas 1 to misclassification). For example, if for a query the minimum error probability is the same for both MSM and ERP, the respective error rates on the training set are the same, and MSM misclassifies the given query while ERP correctly classifies it, then the classification result for this query is considered to be 0.5.

Apart from the classification error rate, we evaluated the *efficiency* of the `Homogeneity-based` method, which is the one that yields at least as good or better classification accuracies than the baseline method on the largest number of datasets compared to the other proposed methods. In particular, we analyzed the *runtimes* for all of its parts, which are reported in Figure 6.4.

The framework and the proposed methods were implemented in Matlab, while DTW, MSM, and ERP were implemented in Java. The experiments were performed on a PC 64-bit running Linux, a Dual-Core AMD Opteron(tm) Processor 8220 SE at 2.8GHz using a single threaded implementation.

6.4.1.4 Training and Test Sets

For several datasets [124] we observed that the error rates on the training and test sets are very different for each of the distance measures, showing that the training set is not representative of the test set. For example, for the training set of the *FaceAll* dataset the error rate for DTW is 6.79%, for MSM 1.07%, and for ERP 2.5%, while for the test set the error rates are much different, i.e., 19.23%, 18.88%, and 20.2%, respectively. For the training set of *OSU* the error rates for DTW and ERP are 33% and 30.5%, while for the test set they are 40.91% and 39.7%, respectively. Another example is the *GunPoint* dataset, for which the error rate of DTW on the training set is 18% and of ERP it is 8%, whereas for the test set it is just 9.33% for DTW and 4% for ERP. The differences are also apparent in the *Fish* dataset, where DTW has error rates 26.29% and 16.57% for the training and test set, MSM has 13.71% and 8%, and ERP achieves 17.14% and 12%.

Since measure selection is about learning statistics for each dataset, we had to be fair on selecting the train and test time series. Thus, based on the iid (independent identically distributed) criterion, for each dataset, all time series originally provided as training and test sets by Keogh et al. [124] were merged, and then for each class half of them were randomly picked and included in the training set while the remaining ones were put in the test set. More formally, assuming that the number of time series of a class is M, $\lfloor M/2 \rfloor$ randomly selected time series comprise the training set, while the rest $M - \lfloor M/2 \rfloor$ objects form the test set. Following this procedure, and, for example, for the aforementioned datasets, the error rates achieved on the training and test sets for all measures are much closer to one another, implying that the training sets are much more representative of the test sets compared to the original split. For the *FaceAll* dataset DTW achieves 3.30% and 3.90% error rates, MSM

1.16% and 1.06%, and ERP 1.52% and 1.86%, and for the *OSU* dataset DTW has error rates 33.18% and 35.14%, and ERP 30% and 31.98% for the training and test set, respectively. With regard to the *GunPoint* dataset, the error rates for DTW are 8% for both training and test sets, and for ERP they are 2% and 3%. Finally, referring to the *Fish* dataset MSM achieves 11.43% and 10.29% error rates for the training and test sets, DTW 25.71% and 23.43%, and ERP 17.71% and 14.29%. The sizes of the two modified sets for each dataset are shown in columns "train size" and "test size" of Table 6.2, respectively.

The datasets used for our experiments are available here:

`http://vlm1.uta.edu/~akotsif/query_sensitive_measure_selection`

6.4.2 Experimental Results

6.4.2.1 Classification Accuracy

The performance of the proposed methods in terms of classification accuracy is shown in Tables 6.3 and 6.4.

The main highlights of our experimental evaluation can be found in Table 6.3, where we show the number of datasets for which our methods provide better, equal, or worse classification error rates than `Cross Validation`. We observe that `Heterogeneity-based` achieves at least as good or better error rates than `Cross Validation` on up to 35 out of 45 datasets. For completeness, we also show the performance of `Basic`. More specifically, for `Distance ratio-based` there are 18 datasets with lower error rates and 11 with the same error rate as that of `Cross Validation`, while for `Homogeneity-based` 23 datasets yield better accuracies than `Cross Validation` and the number of ties is 12.

Table 6.2: Description of the 45 datasets from the UCR repository that were used in our experiments. The table shows for each dataset: the number of training and test objects, the length of each sequence in the dataset, and the number of classes.

ID	Dataset	train size	test size	seq. length	class num.
1	*Synthetic*	300	300	60	6
2	*GunPoint*	100	100	150	2
3	*CBF*	465	465	128	3
4	*FaceAll*	1,122	1,128	131	14
5	*OSU*	220	222	427	6
6	*SwedishLeaf*	555	570	128	15
7	*50Words*	442	463	270	50
8	*Trace*	100	100	275	4
9	*TwoPatterns*	2,499	2,501	128	4
10	*Wafer*	3,582	3,582	152	2
11	*FaceFour*	55	57	350	2
12	*Lightning-2*	60	61	637	2
13	*Lightning-7*	70	73	319	7
14	*ECG*	99	101	96	2
15	*Adiac*	387	394	176	37
16	*Yoga*	1,650	1,650	426	2
17	*Fish*	175	175	463	7
18	*Beef*	30	30	470	5
19	*Coffee*	27	29	286	2
20	*OliveOil*	29	31	570	4
21	*ChlorineConc.*	2,153	2,154	166	3
22	*ECG_torso*	708	712	1,639	4
23	*Cricket_X*	384	396	300	12
24	*Cricket_Y*	384	396	300	12
25	*Cricket_Z*	384	396	300	12
26	*Diatom Red.*	160	162	345	4
27	*ECG5Days*	442	442	136	2
28	*FacesUCR*	1,122	1,128	131	14
29	*Haptics*	231	232	1,092	5
30	*InlineSkate*	324	326	1,882	7
31	*ItalyPower*	547	549	24	2
32	*MALLAT*	1,200	1,200	1,024	8
33	*MedicalImages*	568	573	99	10
34	*MoteStrain*	635	637	84	2
35	*SonySurface*	310	311	70	2
36	*SonySurfaceII*	490	490	65	2
37	*StarLightC.*	4,617	4,619	1,024	3
38	*Symbols*	508	512	398	6
39	*TwoLeadECG*	580	582	82	2
40	*uWaveGest_X*	2,238	2,240	315	8
41	*uWaveGest_Y*	2,238	2,240	315	8
42	*uWaveGest_Z*	2,238	2,240	315	8
43	*WordsSynon.*	450	455	270	25
44	*ECGThorax1*	1,871	1,894	750	42
45	*ECGThorax2*	1,871	1,894	750	42

Table 6.3: Number of datasets for which each method yields lower (*Better*), equal (*Tie*), or higher (*Worse*) classification error rate compared to `Cross Validation`. We observe that `Heterogeneity-based` achieves at least as good or better error rates than `Cross Validation` on up to 35 out of 45 datasets.

	Homogeneity-based	**Distance ratio-based**	**Basic**
Better	23	18	11
Tie	12	11	18
Worse	10	16	16
Better or Tie	**35**	29	29

The classification error rates of `Cross Validation` and the proposed methods are shown in Table 6.4. For each dataset, the classification error rates for MSM, DTW, and ERP for both the training and test sets are also shown, along with the c value used for MSM on that dataset. As mentioned in Section 6.4.1, the value of c that was chosen is the one providing the smallest classification error rate (or highest classification accuracy) for the training set. In cases where more than one values of c provided the same error rate, in Table 6.4 we present all of these values, and we randomly picked the one shown in italics. All rates shown are in percent and the numbers in bold indicate the smallest error rates for each dataset when comparing the two proposed methods and `Cross Validation`. The last column of Table 6.4 ("All Miscl.") presents the number of test time series per dataset that are not classified correctly by any distance measure. This is important since we would like to minimize these numbers, which may happen when adding more distance measures in the pool $\mathcal{L}$. This way our statistical framework may be able to capture the most appropriate measure that classifies correctly each query.

There are several datasets for which the proposed methods provide lower classification error rates for the test set than the `Cross Validation` method. This is because for these datasets our methods are able to select a measure that correctly classifies the test time series for which the measure that is used by `Cross Validation` misclassifies them when using the NN classifier. If the class of the NN is the same for all measures, then the time series are either classified correctly or misclassified whatever measure selected. For the datasets with IDs 13, 37, 40-42, 44, and 45, the classification accuracies given by the `Homogeneity-based` and in all cases but one by the `Distance ratio-based` method are better than the `Cross Validation` method. More specifically, for the *Lightning-7* dataset (ID=13) the error rate of `Cross Validation` is 27.40%, while for `Distance ratio-based` and `Homogeneity-based` it

is 19.18% and 20.55%, respectively. Furthermore, for the *StarLightC.* dataset (ID=37) `Homogeneity-based` and `Distance ratio-based` classify correctly 3 and 29 time series more than `Cross Validation`, respectively. This is because they select the MSM or ERP distance measure instead of the DTW used by the baseline method for these time series. Similarly, for the *uWaveGest_X*, *uWaveGest_Y*, and *uWaveGest_Z* datasets (IDs=40-42), the error rates of the `Homogeneity-based` method are lower than the competitor method that uses MSM (8, 25, and 16 more time series correctly classified by our method for each dataset, respectively). For the last two datasets (IDs=44, 45), the error rates of the `Distance ratio-based` and `Homogeneity-based` methods indicate that there are at least 16 and up to 39 more test time series that are correctly classified than the `Cross Validation` method, which is based on ERP and MSM, respectively. Moreover, for the datasets *Cricket_X*, *Cricket_Y*, and *Cricket_Z* (IDs=23-25), `Cross Validation` achieves error rates 23.99%, 21.97%, and 18.69% by using DTW, while `Distance ratio-based` and `Homogeneity-based` have lower error rates for all of them. In particular, the `Distance ratio-based` has the smallest error rates among all methods for *Cricket_X* (21.21%) and *Cricket_Y* (17.17%), which correspond to 11 and 19 more test time series correctly classified compared to `Cross Validation`, respectively. `Homogeneity-based` has 15.15% error rate for *Cricket_Z*, the lowest among all methods (`Cross Validation` misclassifies 14 more test time series than `Homogeneity-based`). We also present the performance of `Basic`, which achieves lower or equal error rates than `Cross Validation` in 29 datasets. We note that for the *OliveOil* (ID=20) and *WordsSynon.* (ID=43) datasets `Basic` even achieves the lowest error rates among all methods.

For completeness, in Table 6.5 we present the number of datasets for which each of the proposed methods (including `Basic`) perform better, the same, and worse than always selecting one distance measure, i.e., MSM, DTW, or ERP. We observe

that, although MSM is a very competitive distance measure, `Distance ratio-based` and `Homogeneity-based` methods outperform each of the three measures much more often than not. It also has to be mentioned that even `Basic` yields much better classification error rates than both DTW and ERP.

In Table 6.6 we present the probabilities that are the outcome of ANOVA when the input vectors are the classification results of each of the proposed methods against `Cross Validation` (denoted as "C.V.") for all test time series. We also present the probabilities when comparing the classification results of `Basic`, `Distance ratio-based`, and `Homogeneity-based` with each of the distance measures (MSM, DTW, ERP). It has to be mentioned that the lower the probabilities are the more different the vectors of classification results are. When the probabilities are high the vectors are very similar, and when the probabilities are 1 we can conclude that the vectors are several times identical, as it happens, for example, between each of the proposed methods and `Cross Validation` for datasets with IDs 8, 9, and 10, as the corresponding error rates are 0.00%, 0.00%, and 0.28%, respectively. In other words, the same distance measure may be selected (for all test time series) by each of the proposed methods and `Cross Validation`, e.g., for datasets 8 and 10 DTW and MSM is selected, respectively, by the proposed methods and `Cross Validation`. Note that for dataset 9 all measures provide 0.00% error rate in the training and test sets, hence all methods yield the same vectors of classification results.

In conclusion, the number of datasets for which we achieve a lower error rate than `Cross Validation` increases as we move from Scheme 0 to Scheme II. Moreover, based on these results, we can argue that the `Homogeneity-based` method (Scheme II) outperforms the competitor `Cross Validation` method in terms of classification error rate more often than not. This is why we argue about the importance of the intuition to select the most appropriate distance measure for each query in the test

sets, and in addition that these results are worth disseminating to the time series classification community.

6.4.2.2 Runtime

In Figure 6.4, for each dataset, we present the average runtimes of a query for all parts of the `Homogeneity-based` method: MSM, DTW, ERP Computation, selecting the "best" T ("Homogeneity-based scheme"), and determining the most suitable distance measure ("Remaining Time"). Adding up the runtimes of all parts gives the total time, which is also shown.

It can be clearly seen that, for any dataset, the bottleneck of total runtime is the computation of distances for the three measures. DTW takes more time than MSM and ERP for all datasets, which is obvious, e.g., for datasets 22, 30, 32, 37, 44, and 45. The reason for the higher total runtime per query for datasets 16, 22, 29, 30, 32, 37, 40-42, and 44, 45 is the number of their training time series to which the MSM, DTW, and ERP distances have to be evaluated for the query, and also the large length of their time series (as Table 6.2 indicates).

Comparing the proposed methods, since the framework is the same for all of them, the average DTW, MSM, and ERP distance computation times for all queries of each dataset are essentially the same for all methods. In addition, the "Remaining Time" part is negligible for all of them. As a result, the only part that basically differentiates the `Homogeneity-based` method and the rest is the procedure for finding the "best" value for T. However, since the total runtime is dominated by the distance computations, all proposed methods have approximately the same total runtime per query.

Finally, regarding the baseline `Cross Validation` method, the total runtime for a query depends on whether there is a clear distance measure winner in the error

Table 6.4: NN classification error rates attained by the two proposed methods (denoted as `Dist-Ratio` and `Hom.`) as well as `Basic`, and `Cross Validation` on each of the 45 datasets in the UCR repository of time series datasets. In addition, the table shows for each dataset: the classification error rate of MSM, DTW, and ERP for the training and test sets and the value of c used by MSM on that dataset, which yielded the lowest error rate on the training set (when more than one values are given, the one in italics was randomly chosen). All rates are in percent and the numbers in bold indicate the smallest classification error rates for each dataset when comparing the two proposed methods and `Cross Validation`. We also present the error rate of the `Basic` scheme. The number of test time series per dataset that are misclassified by all distance measures is also provided in the last column.

ID	Basic	Dist. Ratio	Hom.	Cross Valid.	train			test			MSM	All Miscl.
					MSM	DTW	ERP	MSM	DTW	ERP	c	
1	1.67	**1.00**	**1.00**	2.00	1.67	1.33	1.00	1.33	1.00	2.00	*0.1*,1	0
2	3.00	3.00	**2.00**	3.00	3.00	8.00	2.00	0.00	8.00	3.00	0.1	0
3	**0.00**	**0.00**	**0.00**	**0.00**	0.43	0.00	0.00	0.22	0.00	0.00	0.1	0
4	1.15	1.15	**1.06**	**1.06**	1.16	3.30	1.52	1.06	3.90	1.86	1	10
5	**19.37**	20.72	**19.37**	**19.37**	15.91	33.18	30.00	19.37	35.14	31.98	0.1	26
6	11.58	11.93	**11.05**	11.58	12.25	21.44	12.97	11.58	20.70	12.63	1	36
7	19.44	**18.36**	20.73	19.22	20.36	35.29	29.86	19.22	33.05	28.51	1	59
8	**0.00**	**0.00**	**0.00**	**0.00**	4.00	0.00	17.00	5.00	0.00	11.00	0.1	0
9	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.00	0.00	0.00	0.00	0.00	1	0
10	**0.28**	**0.28**	**0.28**	**0.28**	0.11	0.50	0.20	0.28	0.78	0.36	1	8
11	11.98	9.65	7.89	**7.60**	1.82	1.82	1.82	3.51	14.04	5.26	0.01,0.1,*1*	2
12	11.48	**8.20**	13.11	14.75	16.67	15.00	13.33	18.03	14.75	14.75	*0.01*,1	2
13	28.08	**19.18**	20.55	27.40	32.86	25.71	32.86	26.03	27.40	34.25	0.01,*1*	10
14	19.80	**16.83**	17.33	**16.83**	9.09	14.14	9.09	13.86	21.78	19.80	1	9
15	38.32	**36.80**	**36.80**	39.34	35.66	39.28	37.73	39.34	38.83	40.86	1	116
16	4.85	5.15	4.97	**4.79**	4.42	8.67	5.58	4.79	8.67	5.88	0.1	49
17	10.86	10.86	**9.71**	10.29	11.43	25.71	17.71	10.29	23.43	14.29	1	13
18	56.67	**53.33**	**53.33**	**53.33**	60.00	63.33	60.00	53.33	60.00	53.33	1	16
19	17.24	20.69	**13.79**	17.24	22.22	22.22	25.93	20.69	13.79	20.69	0.01	2
20	**6.45**	9.68	12.90	12.90	20.69	24.14	24.14	12.90	9.68	12.90	0.01	2
21	7.06	7.15	7.29	**7.01**	7.71	5.67	7.85	7.71	7.01	7.85	0.1,*1*	130
22	**0.14**	**0.14**	**0.14**	**0.14**	0.00	2.54	0.14	0.14	2.67	1.12	1	0
23	23.99	**21.21**	23.74	23.99	22.14	17.97	23.70	24.75	23.99	25.51	1	60
24	21.46	**17.17**	18.94	21.97	22.92	17.45	23.70	19.19	21.97	19.70	1	41
25	18.69	17.68	**15.15**	18.69	27.60	20.57	28.91	22.98	18.69	23.23	1	44
26	**0.62**	**0.62**	**0.62**	**0.62**	0.00	0.63	0.63	0.62	1.23	1.23	1	1
27	**0.68**	0.90	1.02	**0.68**	0.45	1.13	1.13	0.68	1.58	2.26	0.01,*1*	2
28	0.89	0.98	**0.80**	0.89	1.16	3.48	1.87	0.89	3.19	1.24	1	7
29	**52.16**	55.60	**52.16**	**52.16**	55.41	57.14	58.01	52.16	62.50	55.17	1	84
30	43.87	45.09	**42.64**	42.94	45.06	50.93	45.99	42.94	53.99	44.17	1	104
31	3.64	3.73	**3.46**	3.74	4.94	5.67	4.94	3.28	4.92	4.19	1	13
32	**1.17**	**1.17**	**1.17**	1.42	0.10	2.25	0.75	1.42	1.50	1.17	1	4
33	19.37	**18.15**	19.02	19.02	20.60	26.06	24.47	19.02	21.12	21.47	0.1	60
34	3.30	**2.98**	3.14	3.30	2.52	5.51	3.31	3.30	5.49	4.08	0.1	7
35	0.96	0.64	0.96	**0.32**	2.26	4.52	2.90	0.32	1.61	1.61	1	1
36	**0.41**	**0.41**	0.61	**0.41**	1.63	4.29	3.67	0.41	2.45	1.63	1	1
37	6.47	**5.93**	6.49	6.56	8.43	7.06	10.48	8.34	6.56	10.11	0.1	109
38	1.56	1.56	**1.17**	1.37	1.18	1.97	1.77	1.37	1.95	2.34	0.01,*0.1*	4
39	**0.17**	**0.17**	**0.17**	**0.17**	0.00	0.00	0.34	0.17	0.17	0.34	0.01	1
40	21.88	**20.40**	20.67	21.03	21.27	24.66	23.06	21.03	26.52	22.05	1	291
41	28.13	27.90	**26.74**	27.86	28.06	35.17	31.95	27.86	35.18	30.71	1	242
42	27.05	25.85	**25.71**	26.43	27.97	32.98	28.55	26.43	31.03	29.55	1	331
43	**18.46**	19.78	19.78	19.12	19.33	32.67	28.00	19.12	31.43	25.05	1	55
44	16.26	15.73	**15.10**	17.16	19.08	20.36	18.12	18.06	20.91	17.16	1	145
45	10.61	9.87	**9.82**	10.72	10.69	14.27	12.03	10.72	13.73	10.82	1	96

Table 6.5: Number of datasets for which each method yields lower (*Better*), equal (*Tie*), or higher (*Worse*) classification error rate compared to always choosing one distance measure, i.e., MSM, DTW, or ERP.

	Basic			**Distance ratio-based**			**Homogeneity-based**		
	MSM	DTW	ERP	MSM	DTW	ERP	MSM	DTW	ERP
Better	14	35	37	20	37	36	24	37	39
Tie	12	6	5	8	6	3	11	6	5
Worse	19	4	3	17	2	6	10	2	1
Better or Tie	26	41	42	28	43	39	35	43	44

rate on the training set or not. In case one of the three measures provides the smallest classification error rate on the training set, then this measure is evaluated. Thus, given a query of e.g., dataset with ID x, the total runtime is practically given by the point of the curve corresponding to this measure at position x (of the horizontal axis) in Figure 6.4. Similarly, if more than one measures attain the lowest classification error rate, then all these measures contribute to the total runtime for a query, which is the sum of the distance computation times of the query to all training time series for the tied measures. Hence, if, in the worst case, all measures in the pool need to be evaluated for the query, then the total runtime of `Cross Validation` is approximately the same as that of our methods.

An astute reader may argue that the runtime comparison of `Homogeneity-based` against `Cross Validation` is unfair since we could alternatively have used existing speedup methods for DTW [129], or even powerful pruning techniques such as cDTW with the LB_Keogh lower bound [41]. Nonetheless, we argue that any speedup achieved by each method used by `Cross Validation` is also equally beneficial for our framework (and hence for `Homogeneity-based`). This is due to the fact that our framework is using the exact same set of distance measures, and thus any speedup obtained by `Cross Validation` can essentially be exploited by our framework as well.

Table 6.6: The probabilities that are the outcome of the ANOVA statistical test when the input vectors are the classification results (for all test time series) of each of the proposed methods against MSM, DTW, ERP, and `Cross Validation` (denoted as "C.V.") are presented for each of the 45 datasets.

ID	Basic				Dist. Ratio				Hom.			
	MSM	DTW	ERP	C.V.	MSM	DTW	ERP	C.V.	MSM	DTW	ERP	C.V.
1	0.656	1	0.083	0.083	0.656	1	0.083	0.083	0.706	0.480	0.318	0.318
2	0.083	0.058	1	1	0.158	0.033	0.320	0.320	0.083	0.058	1	1
3	0.318	1	1	1	0.318	1	1	1	0.318	1	1	1
4	0.318	0.000	0.021	0.318	1	0.000	0.013	1	0.318	0.000	0.021	0.318
5	0.180	0.000	0.000	0.180	1	0.000	0.000	1	1	0.000	0.000	1
6	0.564	0.000	0.528	0.564	0.257	0.000	0.160	0.257	0.000	0.000	0.377	0.000
7	0.318	0.000	0.000	0.318	0.090	0.000	0.000	0.090	0.318	0.000	0.000	0.318
8	0.025	1	0.001	1	0.025	1	0.001	1	0.025	1	0.001	1
9	1	1	1	1	1	1	1	1	1	1	1	1
10	1	0.000	0.083	1	1	0.000	0.083	1	1	0.000	0.083	1
11	0.018	0.024	0.024	0.022	0.058	0.034	0.083	0.058	0.019	0.164	0.035	0.073
12	0.033	0.159	0.103	0.103	0.321	0.742	0.709	0.709	0.209	0.419	0.419	0.419
13	0.167	0.013	0.004	0.013	0.288	0.024	0.017	0.024	0.683	0.863	0.236	0.863
14	0.158	0.158	0.057	0.108	0.127	0.251	0.227	0.177	0.033	0.508	1	0.517
15	0.025	0.268	0.005	0.025	0.018	0.277	0.003	0.018	0.415	0.778	0.086	0.415
16	0.157	0.000	0.109	0.157	0.318	0.000	0.047	0.318	0.564	0.000	0.027	0.564
17	0.319	0.000	0.083	0.319	0.319	0.000	0.011	0.319	0.319	0.000	0.083	0.319
18	0.000	0.161	0.000	0.000	0.000	0.161	0.000	0.000	0.326	0.326	0.326	0.326
19	1	0.326	1	0.663	0.326	1	0.326	0.663	0.326	0.663	0.573	0.494
20	0.325	1	0.325	0.325	1	0.572	1	1	0.161	0.325	0.161	0.161
21	0.103	0.083	0.047	0.083	0.199	0.034	0.096	0.034	0.039	0.763	0.015	0.763
22	1	0.000	0.008	1	1	0.000	0.008	1	1	0.000	0.008	1
23	0.019	0.041	0.006	0.041	0.528	0.835	0.275	0.835	0.681	1	0.415	1
24	0.249	0.001	0.086	0.001	0.882	0.028	0.640	0.028	0.250	0.564	0.371	0.564
25	0.003	0.372	0.001	0.372	0.000	0.003	0.000	0.003	0.038	1	0.022	1
26	0.000	0.319	0.319	0.000	0.000	0.319	0.319	0.000	0.000	0.319	0.319	0.000
27	0.318	0.180	0.014	0.318	0.180	0.276	0.016	0.180	1	0.045	0.008	1
28	0.318	0.000	0.318	0.318	0.318	0.000	0.096	0.318	1	0.000	0.206	1
29	0.183	0.021	0.862	0.183	1	0.001	0.179	1	1	0.001	0.287	1
30	0.209	0.000	0.675	0.209	0.848	0.000	0.476	0.848	0.565	0.000	0.876	0.565
31	0.318	0.042	0.318	0.318	0.655	0.021	0.158	0.406	0.415	0.071	0.083	0.249
32	0.083	0.414	1.000	0.083	0.083	0.414	1	0.083	0.083	0.414	1	0.083
33	0.370	0.044	0.010	0.370	1.000	0.163	0.052	1	0.528	0.292	0.128	0.528
34	0.480	0.002	0.162	0.480	0.763	0.005	0.201	0.763	1	0.008	0.336	1
35	0.318	0.083	0.180	0.318	0.158	0.158	0.415	0.158	0.158	0.318	0.158	0.158
36	0.000	0.004	0.034	0.000	0.318	0.007	0.096	0.318	0.000	0.004	0.034	0.000
37	0.000	0.013	0.000	0.013	0.000	0.799	0.000	0.799	0.000	0.317	0.000	0.317
38	0.318	0.318	0.249	0.318	0.564	0.045	0.058	0.564	0.318	0.415	0.206	0.318
39	0.000	0.000	0.318	0.000	0.000	0.000	0.318	0.000	0.000	0.000	0.318	0.000
40	0.201	0.000	0.008	0.201	0.530	0.000	0.005	0.530	0.059	0.000	0.775	0.059
41	0.938	0.000	0.000	0.938	0.093	0.000	0.000	0.093	0.109	0.000	0.003	0.109
42	0.339	0.000	0.000	0.339	0.206	0.000	0.000	0.206	0.253	0.000	0.000	0.253
43	0.366	0.000	0.001	0.366	0.318	0.000	0.001	0.318	0.180	0.000	0.000	0.180
44	0.002	0.000	0.028	0.028	0.000	0.000	0.003	0.003	0.016	0.000	0.116	0.116
45	0.046	0.000	0.080	0.046	0.041	0.000	0.056	0.041	0.527	0.000	0.717	0.527

6.5 Conclusions and Future Work

In this chapter we studied the problem of selecting the most appropriate time series distance measure for a given query out of a pool of measures, so as to perform time series NN classification. We demonstrated that the problem is important and chal-

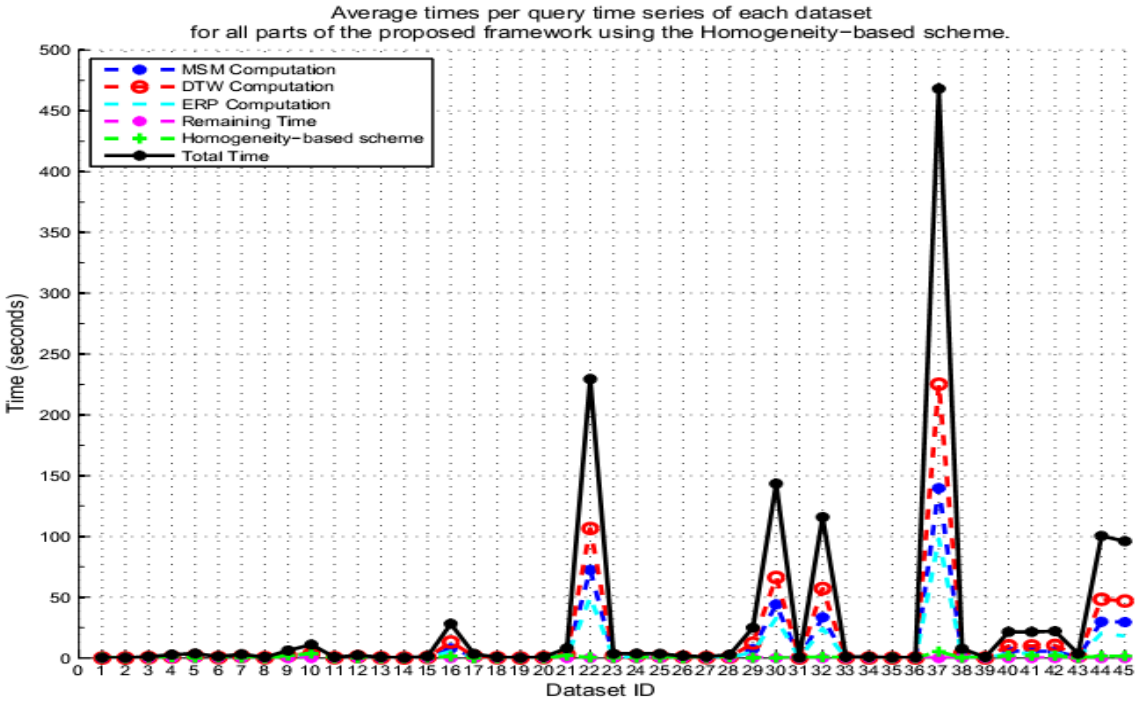


Figure 6.4: Average runtimes per query, for each dataset, for all parts of the Homogeneity-based measure selection method: the computations of MSM, DTW, ERP, Homogeneity-based scheme, and the final measure selection part. The total average runtime per query for each dataset (summing up all of the above parts) is also shown.

lenging, and proposed a novel framework to solve it. The framework consists of two steps, the offline pre-processing step, and the online query classification step. Based on this framework we developed two methods for identifying the T-neighborhood of the query. Each method, in order to create the neighborhood, accounts for a different scheme, named `Distance ratio-based` or `Homogeneity-based`. The classification error rates attained by our framework on 45 datasets are very promising for this problem, since they are at least as good or better than those for `Cross Validation` on up to 35 datasets.

Additional measures can be investigated that will correctly classify the objects misclassified by the current measures in our pool, and that will make our methods

achieve even smaller classification error rates. New schemes can also be explored for defining the T-neighborhood and alternative statistical tests.

CHAPTER 7

MODEL-BASED TIME SERIES INDEXING FOR NEAREST NEIGHBOR CLASSIFICATION

As mentioned in Chapter 6, we can search large time series databases with any of the distance measures referred to so far. However, this approach can be computationally expensive due to the quadratic complexity of the distance measures, unless applying, e.g., lower-bounding techniques. An example of the latter alternative is the lower-bounding technique for cDTW [130], which can achieve important speedups. Another approach would be to first represent each class of the time series database with a *model*, and then perform searching based on the constructed models, which has received limited attention in time series literature. We focus on *Hidden Markov Models* (*HMMs*) [73, 131], which are widely known and have been applied to a variety of domains, such as biology, speech recognition [73], and music retrieval [8, 132]. HMMs model the underlying structure of sequences determining the relationships between their observations [133], and, although they may be computationally expensive for their training, once they are constructed they can be highly applicable to time series, as shown next through the classification task.

In this chapter, we deal with both effectiveness (accuracy) and efficiency (runtime) when searching time series databases, and hence we present a novel approach, named `MTSI` (shorthand for Model-based Time Series Indexing). Given sets of time series of certain classes, `MTSI` first models their underlying structure through the training of one HMM per class. At runtime, given a query time series of unknown class, `MTSI` finds the top-K models that have most likely produced the query. Then,

it refines the search by applying an appropriate distance measure between the query and all the training time series that compose the K selected models. What remains to be answered is what distance measure to use during the refine step. Intuitively, given a collection of distance measures, we can choose the one providing the highest classification accuracy on the training set of the database. The main contributions of this chapter include:

- A novel way of representing time series of a specific class via an HMM, and a comparative evaluation of this representation against three distance measures (DTW, ERP, and MSM) in terms of classification accuracy on the training sets of 45 datasets [124]. The evaluation shows that HMMs can attain significantly higher accuracy in 18 datasets, relatively higher accuracy in 12, and equal accuracy in 4; hence better or equal accuracy in 34 datasets.
- `MTSI`: an indexing framework for effective and efficient time series NN classification. The framework works in a filter-and-refine manner, by exploiting the novel model-based representation of time series belonging to the same class.
- An extensive experimental evaluation on NN classification accuracy between `MTSI` and the `Cross Validation` method defined over DTW, ERP, and MSM, on 33 datasets. We observed that `MTSI` is at least as good as `Cross Validation` in 23 datasets, while achieving a speedup of up to an order of magnitude, showing *both* its effectiveness and efficiency.

We have to mention that although there have been several very interesting time series representation methods proposed in the literature (Section 2.3), they target in representing each individual time series by taking advantage of its structure. However, here we try to represent "groups" of time series that belong to the same class, which is orthogonal to the previously proposed techniques. In addition, our approach differs from selecting the best model in Markovian Processes [134], since at each state we

neither perform an action nor give a reward. Furthermore, model-based kernel for time series analysis requires significant amount of time [135]. Conditional Random Fields (CRFs) [136, 137] can be used for modeling temporal patterns. Nonetheless, we do not target in finding and modeling patterns, rather to represent groups of "homogeneous" sequences by identifying the relationships among their observations.

7.1 MTSI: Model-based Time Series Indexing

Here, we describe how we can represent classes of time series with appropriate training of HMMs, and describe the `MTSI` framework, which takes advantage of the trained models for NN classification. Using the same notation as in Chapter 6, given a distance measure $dist_x$, the distance between time series X and Y is denoted as a function $d_{\text{dist}_x}(X, Y)$.

7.1.1 Hidden Markov Models

When a database consists of sets of time series belonging to certain classes, HMMs can be used to model the different classes after being trained on their respective time series. This lies on the fact that a trained HMM can reflect the probabilistic relations of the values within the sequences, and consequently represent their common structure. Thus, HMMs can be highly applicable for retrieval or classification [132]. Given a query Q, we can look for the model that maximizes the likelihood of having generated Q. With this direction in mind, the time series matching problem is transformed to probabilistic-based matching.

7.1.2 Training HMMs

Assume that we have a dataset $\mathcal{D}$ comprised of N training time series and z classes $C_1, \ldots, C_z$. Let $\mathcal{C}_i$ be the set of training time series that belong to class C_i,

with $i = 1, \ldots, z$. The size of $\mathcal{C}_i$ is denoted as $|\mathcal{C}_i| = n_i$. The training phase of an HMM for each $\mathcal{C}_i$ is split to two phases, which are performed offline: a) initialization and b) iterative refinement.

7.1.2.1 Initialization Step

For each time series X_j $(j = 1, \ldots, n_i)$ of $\mathcal{C}_i$ $(i \in [1, z])$ we compute the average distance of all other time series $X_k \in \mathcal{C}_i$ $(j \neq k)$ to X_j, which we denote as $a^j_{dist_x}$, i.e.,

$$a^j_{dist_x} = \frac{1}{n_i - 1} \sum_{\forall X_k \in \mathcal{C}_i, X_j \neq X_k} d_{dist_x}(X_j, X_k).$$

For the above computation we choose DTW to be the distance measure, i.e., $dist_x =$ DTW, since it has been shown to be one of the most competitive measures for time series matching [32]. In addition, we keep track of the *warping path* of all the pair-wise time series alignments involved in this process.

Next, we identify the *medoid* of $\mathcal{C}_i$, denoted as $X_{\mu_{C_i}}$, which is the time series with the minimum average distance to the rest of the training set, where

$$\mu_{C_i} = \arg\min_j (a^j_{dist_x}), \forall (j = 1, \ldots, n_i).$$

The medoid $X_{\mu_{C_i}}$ is broken into M equal-sized *segments*, where each segment $m \in [1, M]$ corresponds to one HMM state. Using these segments and the stored warping paths we can determine the observed values of each state. Specifically, for each state (that corresponds to a segment m) the observed values include all elements of $X_{\mu_{C_i}}$ in m, along with the elements of all time series in $\mathcal{C}_i$ that have been aligned to elements of m; the latter can be retrieved from the stored warping paths.

A common case in HMMs is to have for each state a Gaussian distribution for E, which is defined by the mean and standard deviation of the stored elements. To compute T, since we consider each segment as a state, at time t when an observation

is emitted we can either stay at the same state or move forward to the next state. Let $|s_t|$ denote the total number of elements at time t of state s_t. The probability of jumping to the next state is $p = n_i/|s_t|$. This is quite straightforward: consider only one segment and one time series X_j ($j = 1, \ldots, n_i$), and suppose that X_j contributes y elements to that segment. Since only the last element in the segment can lead to a transition from s_t to s_{t+1}, the transition probability is $1/y$. Considering now all n_i time series in the segment, the probability of a state transition is $p = n_i/|s_t|$. Finally, the probability of staying at the same state is $(1 - p) = (|s_t| - n_i)/|s_t|$, while for the last state it is 1. In total, the complexity of this step is $O((n_i^2 max_{j\in[1,n_i]}|X_j|)^2)$.

7.1.2.2 Iterative Refinement Step

In this step, we refine the z HMMs constructed during initialization.

For a specific class C_i ($i \in [1, z]$), for each $X_j \in \mathcal{C}_i$, we compute the Viterbi algorithm [73] to find its best state sequence. Specifically, let us denote as δ the Mx$|X_j|$ probability matrix. Each X_j always starts from state 1 (thus $\pi_1 = 1$), and in the initialization phase the log-likelihood of its first element is computed according to the Gaussian distribution. The remaining elements of the first column of δ are set to $-\infty$. Since to get an observation we have either stayed at the same state or have performed a transition from the previous state, in the recursion phase we consider only the values of δ representing the probabilities of the previous element for the previous and the current state. For cell (u, v) these values are $\delta(u, v-1)$ and $\delta(u-1, v-1)$, which were computed in the initialization step. Hence, we first find the most probable transition by computing $m = max(\delta(u, v-1) + \log(t_{uu}), \delta(u-1, v-1) + \log(t_{u-1u}))$, and then $\delta(u, v) = m + \log(e_u(k))$. Finally, we backtrack from $\delta(M, |X_j|)$ and store the elements of X_j falling within each state. Having done this step for all $X_j \in \mathcal{C}_i$,

the mean and standard deviation for E of each state, and also T are updated. The complexity of the aforementioned procedure is $O(M \sum_{j=1}^{n_i} |X_j|)$.

The refinement step is performed for the z HMMs and is repeated until a stopping criterion is met, e.g., the classification accuracy on the N training time series composing $\mathcal{D}$ cannot be further improved (Section 7.2.1.2). The final outcome of this step is a set of z HMMs, denoted as $\mathcal{H} = \{H_1, \ldots, H_z\}$, where each $H_i \in \mathcal{H}$ defines a probability distribution for $\mathcal{C}_i$, which essentially describes the likelihood of observing any time series of class C_i.

7.1.3 Filter-and-Refine Framework

Given $\mathcal{D}$, we are now ready to present the `MTSI` framework.

7.1.3.1 Offline Step

First, we construct $\mathcal{H}$ as described in Section 7.1.2. To make our framework more generic, assume that we have available a set of l distance measures $\{dist_1, \ldots, dist_l\}$. For each $dist_x$ ($x \in [1, l]$) we compute the NN classification accuracy on the N training time series using leave-one-out cross validation.

7.1.3.2 Filter Step

Since we have created a "new" probabilistic space for time series similarity matching, we should define a way of measuring how "good" each HMM model $H_i \in \mathcal{H}$ is for a query Q. This can be achieved by applying the Forward algorithm [73], which computes the likelihood of Q having been produced by H_i. Thus, the "goodness" of H_i is the likelihood estimate given by the algorithm. The complexity of the Forward algorithm is $O(|Q|M^2)$. Note that this step involves only z computations of the Forward algorithm and it is significantly fast, given that in practice z is rarely higher

than 50. This is based on the fact that in the 45 datasets of the UCR archive [124], which cover real application domains, z is at most 50. After computing the likelihood of each H_i for Q, we identify the K models with the highest likelihood of producing Q.

7.1.3.3 Refine Step

Next, the training time series that comprise each of the top-K selected models are evaluated with Q. This evaluation is performed using the distance measure that achieved the highest classification accuracy on the training set during the offline step. Finally, Q is assigned with the class of the closest time series. The complexity of this step is $O(K' comp(dist_x))$, where K' is the total number of training time series corresponding to the K selected models, and $comp(dist_x)$ is the complexity of computing the distance between two time series using $dist_x$ (selected in the offline step).

It has to be mentioned that the smaller the K the faster our approach is. However, since each HMM is a very compact representation of all time series of a certain class, reducing the number of models selected at the filter step may greatly reduce accuracy, as the time series that will be evaluated at the refine step may not include those of the correct class. On the contrary, as K increases towards z, more training time series will be evaluated, resulting in the brute-force approach evaluating Q with all time series when $K = z$, which is certainly undesirable. Hence, a good value for K is needed to achieve a good tradeoff between effectiveness and efficiency. The choice of distance measure also influences accuracy, but it is beyond the scope of this chapter to select the "best" measure for *each* Q.

7.2 Experiments

In this section, we present the setup and the experimental evaluation for HMM-based representation and `MTSI`.

7.2.1 Experimental Setup

7.2.1.1 Datasets

Similarly to Chapter 6, we experimented on the 45 time series datasets available from the UCR archive [124]. In Table 7.1 we present the number of training and test time series ("train size", "test size"), which are different than the ones presented in Table 6.4.1. The length of each time series ("length $|X|$") and the number of classes ("class num. z") for each dataset are also presented.

7.2.1.2 Methods

First, we evaluated the performance of HMMs (Section 7.1.2) against DTW, ERP, and MSM (Section 6.2). Secondly, we compared `MTSI` with `Cross Validation` using the same three measures. The rationales behind selecting these measures are the same as those mentioned in Chapter 6. The `Cross Validation` method works as described in Section 6.4.1. Note that if more than one measures provide the same highest classification accuracy on the training set, then the accuracy of `Cross Validation` is the accuracy on the test set of the measure that outperforms the other tied measure(s) on most datasets (on their training sets).

For each dataset, parameter c of MSM [42] was selected from $\{0.01, 0.1, 1\}$, and was found using the same procedure as described in Section 6.4.1. For the training of HMMs, for each of the 45 datasets, we varied M from 0.1 to $0.9 * |X|$ (step 0.1) and applied 15 refinement iterations (135 combinations). For each combination

we measured the percentage of training time series for which the model producing the highest likelihood through the Forward algorithm was the correct one. Then, the combination leading to the highest accuracy was selected. If more than one combinations provided the same highest accuracy we chose the smallest M (for further ties smallest number of iterations).

7.2.1.3 Evaluation Measures

We first evaluated the performance of HMMs against DTW, ERP, and MSM on the training sets, and between `MTSI` and `Cross Validation` on the test sets in terms of *classification error rate*. For the HMMs, each training time series is evaluated with all HMMs representing the classes of a dataset using the Forward algorithm, and the class of the HMM yielding the highest probability is considered to be the result of the classifier. Secondly, we evaluated the *efficiency* of `MTSI` and `Cross Validation`. Nonetheless, runtime measurements may depend on particular aspects of the hardware, implementation details, compiler optimizations, and programming language. To overcome these limitations, we also present per dataset the percentage of classes selected at the filter step ($(K/z) * 100$), and, more importantly, the percentage of training time series that were finally evaluated by `MTSI`, as the number of time series may (sometimes greatly) deviate among classes in the same dataset. `MTSI` was implemented in Matlab, while, for efficiency, DTW, MSM, ERP, and the Forward algorithm were implemented in Java. Experiments were performed on a PC running Linux, with Intel Xeon Processor at 2.8GHz.

7.2.2 Experimental Results

Next, we present our experimental findings for the methods and evaluation measures described in Section 7.2.1.

7.2.2.1 Classification accuracy of HMMs

In Table 7.1 we show for each of the 45 datasets the classification error rates attained on the training set for MSM, DTW, ERP, and HMMs. The percentage of time series length $|X|$ defining M and the number of iterations to yield the error rate for HMMs is presented in columns "state perc." and "iter. num.", respectively. The c value for MSM is shown in column "parameter c (MSM)". We observe that HMMs achieve better or equal error rate than that of the competitor distance measures in 34 datasets, out of which they outperform them in 30. The performance of HMMs is in many cases significantly better than all competitors. For example, for *ECGFiveDays* HMMs achieve an error rate of 0% as opposed to the next best which is 26.09% (achieved by both ERP and MSM), while for 18 datasets (e.g., *50Words*, *Lightning-7*, *Adiac*, *Beef*, *OliveOil*, and *ECG_torso*) the error rate of HMMs is at least two times lower than that of the competitors. These numbers show that modeling time series classes with HMMs is highly competitive and promising for NN classification.

7.2.2.2 Classification accuracy of MTSI

In Table 7.2 we present the error rates of `MTSI` and `Cross Validation` on the test sets of the 33 datasets with $z > 2$. We have to mention that if $z = 2$ with `MTSI` we can either select one or two models for the refine step. However, $K = 1$ would essentially exclude the refine step, since time series of only one class would be evaluated, which is meaningless. Hence, the classification error rate would depend solely on how well the HMMs represent and discriminate the classes of the dataset, which is not always the case due to their very compact representation of (sometimes large) classes. In addition, $K = 2$ would make our approach perform brute-force search, which is undesirable. In columns "top K", "% classes", "avg train num.",

Table 7.1: NN classification error rates attained by MSM, DTW, ERP, and HMMs on the training set of 45 datasets from the UCR repository of time series datasets. The table shows for each dataset: the number of training and test objects, the length of each time series in the dataset, the number of classes, the value of parameter c used by MSM on that dataset that yielded the lowest error rate on the training set (when two or three values are given, the one in italics was randomly chosen), the number of states as a percentage of the time series length and the number of iterations for which the HMMs achieved the lowest error rate on the training set. The numbers in bold indicate the smallest error rate.

ID	Dataset	train error rate (%)				train	test	length	class	parameter	state	iter.
		MSM	DTW	ERP	HMMs	size	size	$\|X\|$	num. z	c (MSM)	perc.	num.
1	*Synthetic*	1.33	1.00	0.67	**0.33**	300	300	60	6	0.1	0.4	3
2	*CBF*	**0.00**	**0.00**	**0.00**	**0.00**	30	900	128	3	0.1	0.5	2
3	*FaceAll*	**1.07**	6.79	2.50	1.25	560	1,690	131	14	1	0.5	11
4	*OSU*	19.50	33.00	30.50	**17.00**	200	242	427	6	0.1	0.3	11
5	*SwedishLeaf*	12.40	24.60	13.40	**12.20**	500	625	128	15	1	0.9	6
6	*50Words*	21.11	33.11	28.22	**8.89**	450	455	270	50	1	0.5	1
7	*Trace*	1.00	**0.00**	9.00	**0.00**	100	100	275	4	0.01	0.3	2
8	*TwoPatterns*	**0.00**	**0.00**	**0.00**	**0.00**	1,000	4,000	128	4	1	0.1	1
9	*FaceFour*	8.33	25.00	12.50	**0.00**	24	88	350	4	1	0.1	1
10	*Lightning-7*	27.14	32.86	28.57	**7.14**	70	73	319	7	1	0.2	1
11	*Adiac*	38.97	40.51	39.49	**15.90**	390	391	176	37	1	0.5	8
12	*Fish*	13.71	26.29	17.14	**7.43**	175	175	463	7	0.1	0.5	4
13	*Beef*	66.67	53.33	66.67	**23.33**	30	30	470	5	0.1	0.4	2
14	*OliveOil*	16.67	13.33	16.67	**3.33**	30	30	570	4	0.01	0.3	2
15	*ChlorineConc.*	38.97	38.97	**38.76**	56.32	467	3,840	166	3	1	0.7	4
16	*ECG_torso*	12.50	32.50	25.00	**2.50**	40	1,380	1,639	4	1	0.4	3
17	*Cricket_X*	**18.46**	20.26	21.54	25.38	390	390	300	12	1	0.5	14
18	*Cricket_Y*	24.10	20.51	23.33	**19.23**	390	390	300	12	0.1, *1*	0.4	9
19	*Cricket_Z*	24.10	**22.56**	24.87	23.08	390	390	300	12	1	0.4	14
20	*Diatom Red.*	6.25	6.25	6.25	**0.00**	16	306	345	4	0.01, 0.1, *1*	0.1	4
21	*FacesUCR*	2.50	10.00	5.50	**0.50**	200	2,050	131	14	1	0.8	9
22	*Haptics*	49.68	58.71	54.19	**29.68**	155	308	1,092	5	1	0.1	15
23	*InlineSkate*	50.00	59.00	49.00	**43.00**	100	550	1,882	7	1	0.5	3
24	*MALLAT*	5.45	5.45	5.45	**0.00**	55	2,345	1,024	8	1	0.1	1
25	*MedicalImages*	27.82	27.56	**26.51**	34.91	381	760	99	10	0.1	0.4	13
26	*StarLightC.*	10.70	9.60	13.80	**8.60**	1,000	8,236	1,024	3	0.1	0.5	14
27	*Symbols*	**0.00**	4.00	8.00	**0.00**	25	995	398	6	0.1	0.1	1
28	*uWaveGest_X*	25.78	29.35	26.67	**25.22**	896	3,582	315	8	0.1, *1*	0.5	11
29	*uWaveGest_Y*	**28.24**	37.05	33.93	34.60	896	3,582	315	8	1	0.7	13
30	*uWaveGest_Z*	29.24	33.59	31.25	**27.46**	896	3,582	315	8	1	0.2	12
31	*WordsSynon.*	22.10	36.33	28.46	**13.86**	267	638	270	25	1	0.8	13
32	*ECGThorax1*	18.17	20.11	17.83	**7.17**	1,800	1,965	750	42	1	0.5	15
33	*ECGThorax2*	10.83	14.17	11.72	**7.67**	1,800	1,965	750	42	1	0.5	14
34	*Gun Point*	**4.00**	18.00	8.00	8.00	50	150	150	2	0.01	0.3	2
35	*Wafer*	**0.10**	1.40	**0.10**	1.70	1,000	6,164	152	2	1	0.9	4
36	*Lightning-2*	16.67	13.33	13.33	**5.00**	60	61	637	2	0.01	0.1	15
37	*ECG*	14.00	23.00	18.00	**12.00**	100	100	96	2	1	0.8	2
38	*Yoga*	**12.00**	18.33	17.33	22.33	300	3,000	426	2	0.1	0.2	15
39	*Coffee*	25.00	**14.29**	25.00	21.43	28	28	286	2	0.01	0.3	1
40	*ECGFiveDays*	26.09	43.48	26.09	**0.00**	23	861	136	2	1	0.2	4
41	*ItalyPowerDemand*	**4.48**	**4.48**	5.97	5.97	67	1,029	24	2	0.1, *1*	0.9	2
42	*MoteStrain*	15.00	25.00	25.00	**0.00**	20	1,252	84	2	0.1	0.5	7
43	*SonySurfaceI*	10.00	20.00	15.00	**0.00**	20	601	70	2	1	0.1	1
44	*SonySurfaceII*	11.11	14.81	18.52	**3.70**	27	953	65	2	0.1	0.3	1
45	*TwoLeadECG*	4.35	8.70	4.35	**0.00**	23	1,139	82	2	*0.01*, 0.1	0.1	4

"% train obj." we show the K value used at the filter step of `MTSI` (due to space limitations we present the smallest $K < z$ for which the accuracy could not be further improved or provided a competitive error rate), the ratio $(K/z) * 100$, the

average number of training time series evaluated per query at the refine step, and the percentage of the "train size" to which this average corresponds to, respectively.

We observe that `MTSI` achieves at least as good or better error rates than `Cross Validation` in 23 datasets; it is better in 17 datasets and equal in 6. There are two reasons for such competitive performance of `MTSI`: a) the correct class of the test time series is among the K models selected at the filter step, and the distance measure applied at the refine step is able to better differentiate the correct class from the rest $K-1$ classes, since there are less training objects to throw away as being "bad" matches compared to brute-force search, and b) the HMMs of these 23 datasets have been constructed exploiting a sufficient number of training time series comprising their classes, which makes the probability distribution of such classes effectively represent these (and similar) time series.

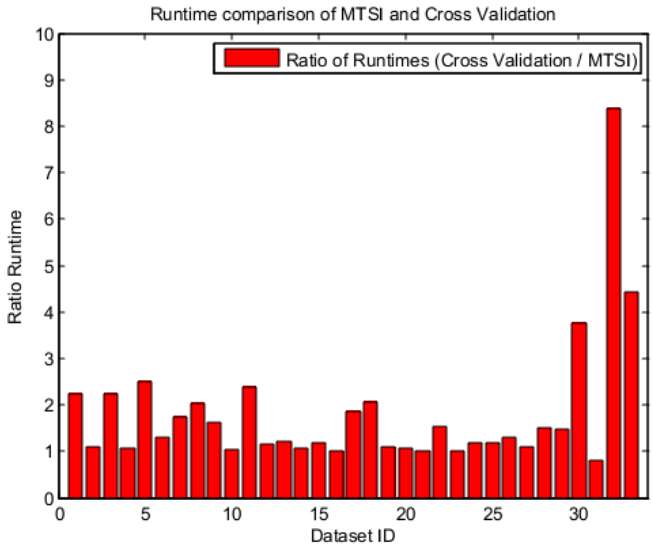


Figure 7.1: Speedup of `MTSI` against `Cross Validation` for 33 datasets. Each bar represents the ratio of the `Cross Validation` average total runtime to that of `MTSI` for NN classification of a test object.

Table 7.2: NN classification error rates attained by `MTSI` and `Cross Validation` on the test set of 33 datasets from the UCR repository of time series datasets. The table also shows for each dataset: the classification error rate of MSM, DTW, and ERP on the test set, the number of HMM models used at the refine step of `MTSI` and the respective percentage of classes it corresponds to, the average number of training objects evaluated at the refine step per test object, and the percentage of training objects this average corresponds to. The numbers in bold indicate the smallest error rate.

ID	Dataset	MTSI	Cross Valid.	error rate (%) MSM	DTW	ERP	top K	% classes	avg train num.	% train obj.
1	*Synthetic*	**3.33**	3.70	2.67	0.70	3.70	2	33.33	100.00	33.33
2	*CBF*	3.67	**1.22**	1.22	0.30	0.30	2	66.67	20.45	68.16
3	*FaceAll*	18.99	**18.88**	18.88	19.20	20.20	5	35.71	200.00	35.71
4	*OSU*	22.73	**19.83**	19.83	40.90	39.70	5	83.33	169.00	84.50
5	*SwedishLeaf*	**9.60**	10.40	10.40	21.00	12.00	3	20.00	100.87	20.17
6	*50Words*	**19.56**	**19.56**	19.56	31.00	28.10	40	80.00	412.26	91.61
7	*Trace*	**0.00**	**0.00**	7.00	0.00	17.00	2	50.00	49.82	49.82
8	*TwoPatterns*	**0.05**	0.08	0.08	0.00	0.00	2	50.00	498.47	49.85
9	*FaceFour*	**4.55**	5.68	5.68	17.00	10.20	2	50.00	11.44	47.68
10	*Lightning-7*	**21.92**	23.29	23.29	27.40	30.10	6	85.71	64.18	91.68
11	*Adiac*	**33.76**	38.36	38.36	39.60	37.90	2	5.41	20.96	5.37
12	*Fish*	**7.43**	8.00	8.00	16.70	12.00	6	85.71	149.87	85.64
13	*Beef*	**46.67**	50.00	50.00	50.00	50.00	3	60.00	18.00	60.00
14	*OliveOil*	16.67	**13.33**	16.67	13.33	16.67	3	75.00	21.80	72.67
15	*ChlorineConc.*	40.42	**37.40**	37.27	35.20	37.40	2	66.67	362.14	77.55
16	*ECG_torso*	15.07	**10.29**	10.29	34.90	25.00	3	75.00	30.36	75.91
17	*Cricket_X*	**25.90**	27.18	27.18	22.30	29.23	5	41.67	162.54	41.68
18	*Cricket_Y*	**20.00**	20.80	16.67	20.80	21.28	5	41.67	162.38	41.64
19	*Cricket_Z*	**20.77**	**20.77**	21.54	20.77	24.36	10	83.33	330.00	84.62
20	*Diatom Red.*	**4.58**	**4.58**	4.58	3.30	5.23	3	75.00	14.67	91.67
21	*FacesUCR*	**3.20**	3.27	3.27	9.51	4.24	13	92.86	191.02	95.51
22	*Haptics*	**57.47**	59.42	59.42	62.30	57.47	3	60.00	98.00	63.22
23	*InlineSkate*	57.45	**56.91**	55.64	61.60	56.91	6	85.71	87.25	87.25
24	*MALLAT*	**6.74**	**6.74**	6.74	6.60	7.46	6	75.00	41.98	76.33
25	*MedicalImages*	**27.89**	**27.89**	24.74	26.30	27.89	7	70.00	334.98	87.92
26	*StarLightC.*	9.35	**9.30**	11.72	9.30	13.62	2	66.67	759.62	75.96
27	*Symbols*	3.12	**3.02**	3.02	5.00	5.83	5	83.33	21.00	83.98
28	*uWaveGest_X*	**22.28**	22.36	22.36	27.30	25.71	5	62.50	574.04	64.07
29	*uWaveGest_Y*	**30.35**	30.37	30.37	36.60	33.61	5	62.50	553.80	61.81
30	*uWaveGest_Z*	**29.12**	31.07	31.07	34.20	32.97	2	25.00	222.04	24.78
31	*WordsSynon.*	23.67	**23.51**	23.51	35.10	32.13	24	96.00	263.92	98.85
32	*ECGThorax1*	**18.37**	19.29	18.27	20.90	19.29	2	4.76	85.99	4.78
33	*ECGThorax2*	**10.89**	11.25	11.25	13.50	10.74	7	16.67	300.03	16.67

Carefully analyzing our experimental findings, we concluded that 16 training time series is a sufficient number to provide a good model representing a class of objects. This claim is supported by the following examples, where `MTSI` yields worse error rates than `Cross Validation`. Datasets 14, 16, and 20 consist of 4 classes, but no class of the first two has more than 15 training time series, while the classes of the latter include only 1, 6, 5, and 4 time series. Moreover, none of the 6 classes of dataset with ID 27 has more than 8 time series, *WordsSynon.* (ID 31) with $z = 25$ has only 4 classes with more than 15 time series, as happens with 3 out of the 7 classes

of *InlineSkate* (ID 23). The error rates for datasets *ChlorineConc.* and *StarLightC.* (ID 15 and 26) can be attributed to overfitting, since for the first no class comprises of less than 91 time series, while for the second all classes have more than 152 time series. In addition, building a histogram over the number of classes that include specific numbers of training time series, we observed that 167 out of the 399 classes (comprising the 33 datasets) have up to 15 training time series. As a result, we would like to emphasize the need for a sufficient number of training time series per class.

7.2.2.3 Efficiency

In Figure 7.1 we present the average *speedup* per test time series when using `MTSI` instead of `Cross Validation` for 33 datasets. The speedup is the runtime ratio of `Cross Validation` over `MTSI`, and it intuitively depends on K. For example, we gain up to an order of magnitude in terms of runtime for dataset with ID 32, since `MTSI` selects only 2 out of 42 classes at the filter step, and its error rate is lower than that of `Cross Validation`. We observe that there are several datasets for which there is no significant speedup. This is mainly because, for these datasets, `MTSI` could not achieve a competitive error rate for large K, even for $z - 1$ in some cases. Thus, for such values its runtime converged to that of brute-force using the appropriate distance measure. Additionally, there may be cases where the length of the time series is not huge enough to provide a noticeable difference in the runtimes of the two competitors (ID 25), resulting in a slight speedup. The latter result may also happen when "train size" is small and/or the average number of training time series for the K selected models is much higher than that of the non-selected ones. The last claim holds, e.g., for datasets with ID 6, 15, 20, 25, 26, where the value of "% train obj." is significantly higher than that of "% classes", showing that the training time series are not equally distributed to all classes. In Table 7.2 the percentage of training time

series evaluated ranges from just 4.78% (ID 32) to 98.85% (ID 31), which is the worst possible case since no smaller K could provide better accuracy. We have to point out, though, that out of the 23 datasets for which `MTSI` is better than or equal to `Cross Validation` there are 11 datasets for which less than 50% of their training set is evaluated.

Based on these results, we can argue that `MTSI` outperforms `Cross Validation` in classification error rate more often than not, while allowing for a speedup of up to an order of magnitude. An acute reader may argue that the runtime comparison of the two methods is unfair since we could alternatively have used existing speedup methods for DTW[129], or even faster techniques such as cDTW with LB_Keogh [130]. Nonetheless, we argue that any speedup achieved by each method used by `Cross Validation` is also equally beneficial for `MTSI`. This is due to the fact that `MTSI` is using the exact same set of methods for the refine step, and thus any speedup obtained by `Cross Validation` is essentially exploited by `MTSI` as well (the filter step cost is negligible compared to the refine step).

7.3 Conclusions and Future Work

In this chapter, we presented an effective way of modeling classes of time series via HMMs, which have been shown to be highly applicable to a variety of domains. Based on such models, we presented `MTSI`, a filter-and-refine framework for NN classification of time series, which, given a query time series, first selects the K most probable models having produced the query, and then applies a distance measure between the query and the training time series of the K classes. Experimenting with 45 widely known time series datasets and three distance measures, DTW, ERP, and MSM, we observed that HMMs provide better or equal classification accuracies than

the competitor measures on the training set in 34 datasets. Moreover, `MTSI` has equal or better accuracy than `Cross Validation` in 23 out of 33 datasets, while achieving a speedup of up to an order of magnitude.

For future work `MTSI` can be applied on larger datasets with more classes, where we expect the performance of our approach to further improve. Individually training each HMM, instead of forcing the same number of states and iterations for all HMMs in a dataset, will also be very beneficial. Finally, the decision for K could be automated so as to provide a satisfying tradeoff between accuracy and speedup, and also incorporate `MTSI` in semi-supervised learning methods.

CHAPTER 8

CASE STUDY: MODEL-BASED VS. DISTANCE-BASED SEARCH IN TIME SERIES DATABASES

As mentioned in Chapter 6, large databases of time series can be exploited so as to extract knowledge on what has happened in the past or to recognize what is happening in the present. Assume that we want to *search* the database for time series corresponding to a specific activity, such as "person falling down". The search process must decide, for each time series, whether that time series is a good match for this particular search (of instances of "person falling down") or not. This can be done as follows: (a) assign a score to each time series in the database, which indicates how good a match each time series is for the particular search we are conducting, (b) rank database time series according to their score, (c) return to the user the top-K matches, where K is a user-specified parameter.

The most critical step of the aforementioned procedure is the first one. Any mathematically valid scoring function can be used, but the choice will essentially determine the accuracy of the results. Thus, it is of particular importance to assign the best scores to the time series that the user would consider to be "correct matches" for the query.

In distance-based search, to perform our search we provide as a query a time series, e.g., time series corresponding to the "falling down" activity. Then, the score assigned to each database time series is the distance/similarity score that is computed using the selected measure. In model-based search, we specify what we are looking for by submitting as a query a model of the activity. This model can be a Hidden Markov

Model, which has been trained on time series, e.g., of the "falling down" activity. The score that is assigned to each database time series is the output of the model on that time series. For HMMs, this score can be computed through the Forward algorithm [73] taking as input the HMM and the time series.

The main contributions of this chapter include: (a) a comparative evaluation of representing classes of time series via HMMs against four distance measures, DTW, cDTW, ERP, and MSM in terms of classification accuracy on the training sets of 45 datasets [124]. The evaluation shows that HMMs can attain significantly higher accuracy in 17 datasets, relatively higher accuracy in 11, and equal accuracy in 4; hence better or equal accuracy in 32 datasets. (b) An extensive experimental evaluation of model-based search with HMMs and distance-based search with DTW, cDTW, ERP, and MSM, on 45 datasets in terms of precision vs. recall and runtime. We observed that HMMs can produce significantly better tradeoffs than the competitors when trained with a sufficient number of training time series, while they are usually slower than the distance-based methods [138].

We should mention that the procedure for training HMMs so as to represent classes of time series is the same as the one described in Section 7.1.2.

8.1 Experiments

8.1.1 Experimental Setup

In this section, we present the setup for the model-based and distance-based time series search comparison.

We experimented on the 45 time series datasets available from the UCR archive [124], which were also used in Chapters 6 and 7. We compared HMMs with four distance measures. Although any measure (Section 2.4) can be used to perform

time series similarity search, an exhaustive consideration of all distance measures is practically impossible and beyond the scope of this chapter. We investigated DTW, cDTW with Sakoe-Chiba band [37], ERP, and MSM. The rationales for selecting DTW, ERP, and MSM have been mentioned in Section 6.2. The reason for also examining the behavior of cDTW is because it is a much faster version of DTW providing equally good or better results than DTW [32]. The time complexity of all measures is quadratic, except for cDTW, which depends on the band constraint value. The performance of HMMs was evaluated against DTW, cDTW, ERP, and MSM on the training sets in terms of *classification error rate*. To evaluate the model and distance-based time series search methods we used *precision* and *recall* for accuracy, and *runtime* for efficiency [138].

With regard to the distance-based search methods, for each dataset, for each class we used all of its training time series as queries (for a total of 15,741 queries for all datasets and classes), and found the DTW, cDTW, ERP, and MSM of all test time series to these queries (in total 61,691 test time series for all datasets). Regarding the model-based search method, we used each of the 423 trained models as a query and computed, for each dataset, the probability of each time series of the test set being produced by that query model. For both types of search methods, after computing the distances/probabilities, we sorted them in ascending/descending order and identified the ranks of the test time series that belong to the same "correct" class with that represented by each query. This was done in order to be able to compute the precision and recall. We note that for the distance-based methods, for each class, since there are many query time series we took the average precision and recall [138]. All distance measures and the Forward algorithm were implemented in Java. Experiments were performed on a PC running Linux, with Intel Xeon Processor at 2.8GHz.

8.1.2 Experimental Results

Next, we present our experimental findings.

8.1.2.1 Classification accuracy of HMMs

In Table 8.1 we show for each of the 45 datasets the classification error rates attained on the training set for MSM, DTW, cDTW, ERP, and HMMs. The percentage of time series length $|X|$ defining M and the number of iterations to yield the error rate for HMMs, along with the value of the c parameter for MSM are also presented (all these values are the same as the ones presented in Table 7.1). We observe that HMMs achieve better or equal error rate than that of the competitor distance measures in 32 datasets, out of which they outperform them in 28. The performance of HMMs is in many cases significantly better than all competitors. For example, for *ECGFiveDays* HMMs achieve an error rate of 0% as opposed to the next best which is 17.39%, while for 18 datasets (e.g., *50Words*, *Lightning-7*, *Adiac*, *Beef*, *OliveOil*, and *ECG_torso*) the error rate of HMMs is at least two times lower than that of the competitors. These numbers show that modeling time series classes with HMMs is highly competitive and promising for searching time series databases [138].

8.1.2.2 Precision vs. Recall

In Figures 8.1 - 8.8, we present the average precision vs. recall of all classes (out of 423), which have at least 1, 10, 60, 120, and 200, less than 5 and 10, and between 10 and 15 training time series [138]. In all figures we also show the number of classes of all datasets that satisfy the aforementioned thresholds ("num selected classes"), and also the total number of training time series of these selected classes

Table 8.1: NN classification error rates attained by MSM, DTW, cDTW, ERP, and HMMs on the training set of 45 datasets from the UCR repository of time series datasets. The table shows for each dataset: the number of training and test objects, the length of each time series in the dataset, the number of classes, the value of parameter c used by MSM on that dataset that yielded the lowest error rate on the training set (when two or three values are given, the one in italics was randomly chosen), the number of states as a percentage of the time series length and the number of iterations for which the HMMs achieved the lowest error rate on the training set. The numbers in bold indicate the smallest error rate.

ID	Dataset	train error rate (%)					train	test	length	class	parameter	state	iter.
		MSM	DTW	cDTW	ERP	HMMs	size	size	$\|X\|$	num. z	c (MSM)	perc.	num.
1	*Synthetic*	1.33	1.00	**0.33**	0.67	**0.33**	300	300	60	6	0.1	0.4	3
2	*CBF*	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	30	900	128	3	0.1	0.5	2
3	*FaceAll*	**1.07**	6.79	4.64	2.50	1.25	560	1,690	131	14	1	0.5	11
4	*OSU*	19.50	33.00	25.00	30.50	**17.00**	200	242	427	6	0.1	0.3	11
5	*SwedishLeaf*	12.40	24.60	18.00	13.40	**12.20**	500	625	128	15	1	0.9	6
6	*50Words*	21.11	33.11	23.33	28.22	**8.89**	450	455	270	50	1	0.5	1
7	*Trace*	1.00	**0.00**	1.00	9.00	**0.00**	100	100	275	4	0.01	0.3	2
8	*TwoPatterns*	**0.00**	**0.00**	0.20	**0.00**	**0.00**	1,000	4,000	128	4	1	0.1	1
9	*FaceFour*	8.33	25.00	12.50	12.50	**0.00**	24	88	350	4	1	0.1	1
10	*Lightning-7*	27.14	32.86	20.00	28.57	**7.14**	70	73	319	7	1	0.2	1
11	*Adiac*	38.97	40.51	39.74	39.49	**15.90**	390	391	176	37	1	0.5	8
12	*Fish*	13.71	26.29	22.86	17.14	**7.43**	175	175	463	7	0.1	0.5	4
13	*Beef*	66.67	53.33	50.00	66.67	**23.33**	30	30	470	5	0.1	0.4	2
14	*OliveOil*	16.67	13.33	10.00	16.67	**3.33**	30	30	570	4	0.01	0.3	2
15	*ChlorineConc.*	38.97	38.97	**36.62**	38.76	56.32	467	3,840	166	3	1	0.7	4
16	*ECG_torso*	12.50	32.50	7.50	25.00	**2.50**	40	1,380	1,639	4	1	0.4	3
17	*Cricket_X*	18.46	20.26	**17.18**	21.54	25.38	390	390	300	12	1	0.5	14
18	*Cricket_Y*	24.10	20.51	**18.21**	23.33	19.23	390	390	300	12	0.1, *1*	0.4	9
19	*Cricket_Z*	24.10	22.56	**17.69**	24.87	23.08	390	390	300	12	1	0.4	14
20	*Diatom Red.*	6.25	6.25	6.25	6.25	**0.00**	16	306	345	4	0.01, 0.1, *1*	0.1	4
21	*FacesUCR*	2.50	10.00	8.00	5.50	**0.50**	200	2,050	131	14	1	0.8	9
22	*Haptics*	49.68	58.71	46.45	54.19	**29.68**	155	308	1,092	5	1	0.1	15
23	*InlineSkate*	50.00	59.00	58.00	49.00	**43.00**	100	550	1,882	7	1	0.5	3
24	*MALLAT*	5.45	5.45	1.82	5.45	**0.00**	55	2,345	1,024	8	1	0.1	1
25	*MedicalImages*	27.82	27.56	**26.25**	26.51	34.91	381	760	99	10	0.1	0.4	13
26	*StarLightC.*	10.70	9.60	9.30	13.80	**8.60**	1,000	8,236	1,024	3	0.1	0.5	14
27	*Symbols*	**0.00**	4.00	4.00	8.00	**0.00**	25	995	398	6	0.1	0.1	1
28	*uWaveGest_X*	25.78	29.35	**24.89**	26.67	25.22	896	3,582	315	8	0.1, *1*	0.5	11
29	*uWaveGest_Y*	28.24	37.05	**27.57**	33.93	34.60	896	3,582	315	8	1	0.7	13
30	*uWaveGest_Z*	29.24	33.59	30.13	31.25	**27.46**	896	3,582	315	8	1	0.2	12
31	*WordsSynon.*	22.10	36.33	27.34	28.46	**13.86**	267	638	270	25	1	0.8	13
32	*ECGThorax1*	18.17	20.11	18.11	17.83	**7.17**	1,800	1,965	750	42	1	0.5	15
33	*ECGThorax2*	10.83	14.17	12.44	11.72	**7.67**	1,800	1,965	750	42	1	0.5	14
34	*Gun Point*	**4.00**	18.00	**4.00**	8.00	8.00	50	150	150	2	0.01	0.3	2
35	*Wafer*	**0.10**	1.40	0.30	**0.10**	1.70	1,000	6,164	152	2	1	0.9	4
36	*Lightning-2*	16.67	13.33	10.00	13.33	**5.00**	60	61	637	2	0.01	0.1	15
37	*ECG*	14.00	23.00	14.00	18.00	**12.00**	100	100	96	2	1	0.8	2
38	*Yoga*	**12.00**	18.33	17.67	17.33	22.33	300	3,000	426	2	0.1	0.2	15
39	*Coffee*	25.00	**14.29**	**14.29**	25.00	21.43	28	28	286	2	0.01	0.3	1
40	*ECGFiveDays*	26.09	43.48	17.39	26.09	**0.00**	23	861	136	2	1	0.2	4
41	*ItalyPowerDemand*	**4.48**	**4.48**	**4.48**	5.97	5.97	67	1,029	24	2	0.1, *1*	0.9	2
42	*MoteStrain*	15.00	25.00	25.00	25.00	**0.00**	20	1,252	84	2	0.1	0.5	7
43	*SonySurfaceI*	10.00	20.00	10.00	15.00	**0.00**	20	601	70	2	1	0.1	1
44	*SonySurfaceII*	11.11	14.81	14.81	18.52	**3.70**	27	953	65	2	0.1	0.3	1
45	*TwoLeadECG*	4.35	8.70	8.70	4.35	**0.00**	23	1,139	82	2	*0.01*, 0.1	0.1	4

("sum training t.s."). DTW is referred to as "Unconstrained DTW" and cDTW as "Constrained DTW".

In Figures 8.1 (i.e., all 423 classes have been considered in the displayed curves) and 8.2 we observe that the precision-recall curve for model-based search is not worse

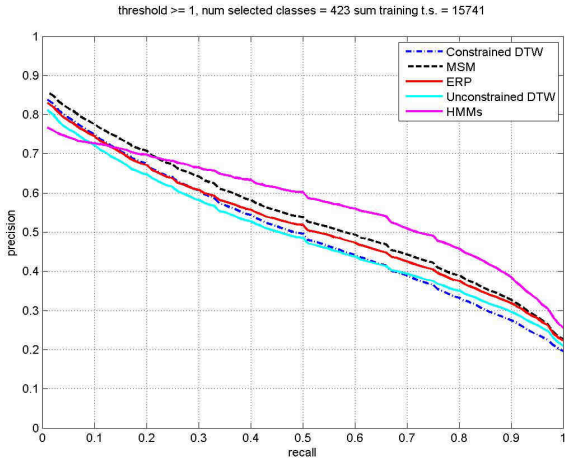


Figure 8.1: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is >= 1.

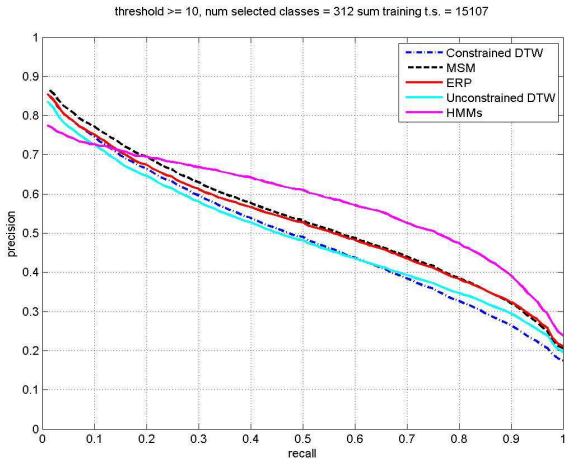


Figure 8.2: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is >= 10.

than all other distance-based methods in 9% recall, while it is better than all competitors in 20% recall. In Figure 8.3 HMMs method is best in 5% recall, showing that when there are many time series to train the HMMs, the model-based method performs noticeably better than the distance-based search methods. On the contrary, when the number of training time series gets extremely high, i.e., at least 120 or (much worse) 200 (Figures 8.4 and 8.5), the model-based method is not better than

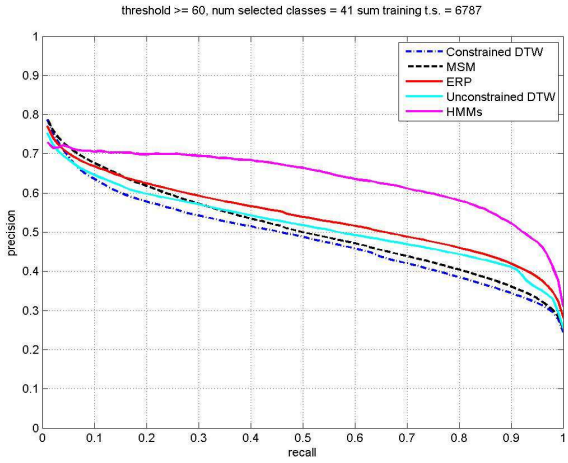


Figure 8.3: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is >= 60.

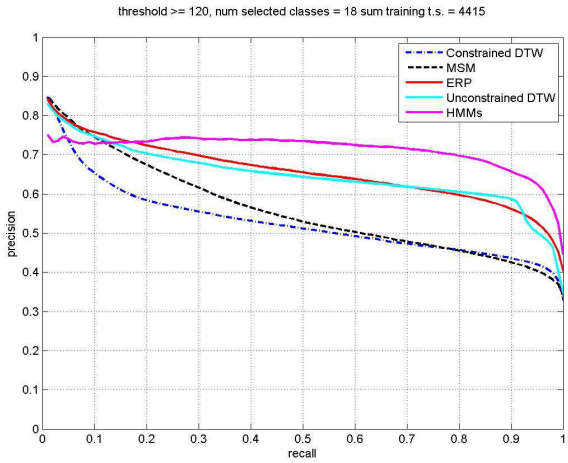


Figure 8.4: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is >= 120.

the competitors until when recall is around 17% and 33%, respectively. In the latter figure we also observe that the curves for DTW and ERP-based methods are very close to that of model-based. The main reason for the last effect is that the curve for HMMs includes the results from classes with more than 300 training time series, i.e., the third class of dataset 26 that has 573 and the second class of dataset 35 that has 903 training time series - 1,476 in total, consisting the 9.37% of the total number of

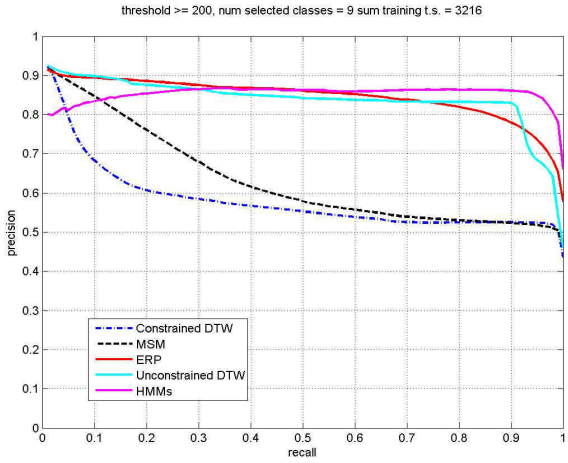


Figure 8.5: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is >= 200.

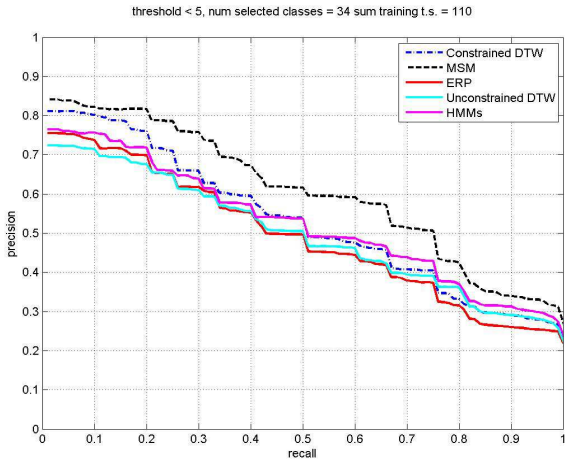


Figure 8.6: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is < 5.

training time series of all datasets). As a result, there has been overfitting for these HMMs, which influences their performance when searching time series datasets. However, it is important to note that, as the threshold increases (Figures 8.1 - 8.5), the precision is getting higher as the recall increases, which is essentially what we target for when searching time series databases. Referring to the distance-based methods, in Figures 8.1 and 8.2 the worst performance is presented by DTW and cDTW, while

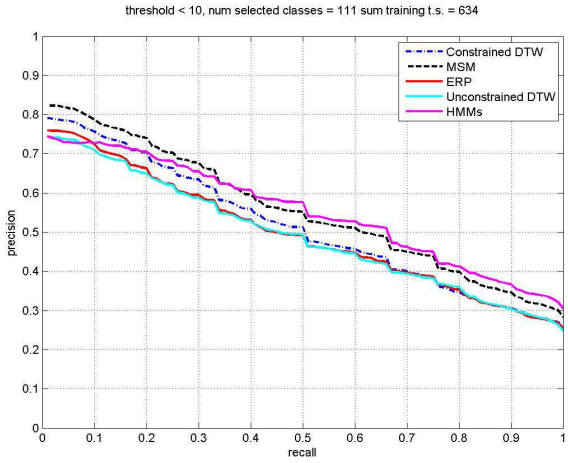


Figure 8.7: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is < 10.

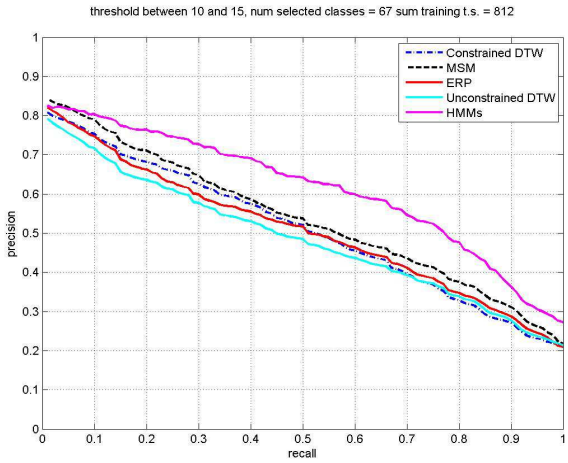


Figure 8.8: Precision vs. recall for model and distance-based time series search, when the number of training time series of the classes is between 10 and 15.

in Figures 8.3, 8.4, and 8.5, cDTW and MSM present the worst curve among all methods.

In Figures 8.6 and 8.7 we show that the precision-recall curve for HMMs is particularly influenced by the number of time series used to train them. In Figure 8.6 the curve corresponds to the average precision vs. recall of all classes/models that have less than 5 training time series, while including classes with more training

Table 8.2: Average time (in seconds) of DTW, cDTW, MSM, ERP, and HMMs for searching on each of the 45 datasets from the UCR repository of time series datasets.

ID	DTW	cDTW	MSM	ERP	HMMs
1	0.102	0.03	0.09	0.048	0.222
2	1.236	0.414	1.08	0.552	3.534
3	2.466	0.396	2.148	1.074	6.888
4	3.438	0.786	2.724	1.506	9.648
5	0.828	0.126	0.696	0.366	6.678
6	2.568	0.564	2.076	1.134	7.74
7	0.6	0.096	0.492	0.258	1.08
8	5.586	1.068	4.62	2.472	3.018
9	0.864	0.12	0.708	0.372	0.504
10	0.588	0.114	0.498	0.258	0.684
11	0.954	0.156	0.75	0.42	2.856
12	2.844	0.504	2.19	1.26	8.736
13	0.51	0.048	0.408	0.24	1.29
14	0.768	0.084	0.612	0.336	1.35
15	9.042	0.894	7.848	3.9	58.074
16	343.44	77.646	283.536	187.506	732.216
17	2.754	0.666	2.388	1.218	8.388
18	2.79	1.164	2.376	1.2	6.864
19	2.79	0.654	2.382	1.206	6.864
20	2.868	0.27	2.232	1.206	1.65
21	3.06	1.008	2.634	1.32	13.722
22	30.246	5.394	23.814	13.446	16.488
23	179.946	81.78	152.154	99.09	483.348
24	202.77	23.928	158.07	88.728	111.804
25	0.708	0.306	0.516	0.276	1.446
26	717.522	295.884	544.434	322.932	2082.708
27	12.33	3.03	9.396	5.1	7.068
28	27.864	4.668	20.874	11.526	84.786
29	27.906	4.722	21.06	11.586	118.428
30	27.84	5.652	20.88	11.556	33.024
31	3.648	0.894	2.922	1.53	18.156
32	89.34	11.346	70.728	38.568	257.526
33	89.166	11.478	71.064	38.946	259.422
34	0.264	0.03	0.21	0.12	0.174
35	11.382	1.404	8.88	4.992	62.634
36	1.998	0.438	1.614	0.87	1.134
37	0.078	0.012	0.066	0.036	0.384
38	42.756	5.502	33.132	18.69	50.4
39	0.18	0.03	0.15	0.084	0.336
40	1.362	0.15	1.11	0.57	1.518
41	0.084	0.042	0.078	0.054	0.312
42	0.75	0.114	0.63	0.342	2.19
43	0.27	0.048	0.234	0.12	0.168
44	0.492	0.066	0.33	0.174	0.606
45	0.696	0.162	0.522	0.294	0.402

time series, up to 10 for Figure 8.7, the curve is much better. In both figures the most competitive distance-based method is MSM. Finally, in Figure 8.8 we present the average precision vs. recall of all methods for all classes that have at least 10 and up to 15 training time series. Clearly, we can see that the HMMs perform best for any recall value, showing that when the HMMs have been constructed exploiting a sufficient (and concurrently not overwhelming) number of training time series comprising their classes, the probability distribution of such classes effectively represents their (and similar) time series.

8.1.2.3 Efficiency

In Table 8.2 for the distance-based methods we present, for each dataset, the average time to find the distance of a training time series/query with all test time series, and for the model-based method we present the average time per model to compute the Forward algorithm with all test time series [138]. According to Table 8.2, cDTW method is always faster than the competitor methods, except for dataset with ID 25 where ERP is the fastest method. We also observe that ERP is 1.5 to 2 time faster than MSM, which is always faster than DTW. In addition, model-based search is basically slower than all competitors, except for datasets with ID 8 (it is better than DTW), 9 (worse than only cDTW), and 20, 22, 24, 27, 36, 43, and 45 (worse than only cDTW and ERP). The explanation for these numbers is that the Forward algorithm is more time consuming than the distance measures. We also have to note that in several datasets the differences in runtimes are not clearly noticeable among the competitors.

8.2 Conclusions and Future Work

In this chapter, we compared the classification error rates of HMMs and four distance measures, DTW, cDTW, MSM, and ERP, on the training sets of 45 datasets. We observed that HMMs achieve better or equal error rates than the competitors on 32 datasets. Furthermore, evaluating the model-based search with the four distance-based methods on the 45 datasets we noticed that the model-based method provides better tradeoffs between precision and recall than the competitors, especially when a sufficient number of time series is provided to train the HMMs. For future work, to further improve the precision-recall curves, each HMM (representing a class of a database) can be individually trained with a different number of states and iterations,

instead of forcing all HMMs of a database to have the same values for states and iterations. In addition, model-based search can be made more efficient by exploring new ways of computing the Forward algorithm faster.

CHAPTER 9

IBSM: INTERVAL-BASED SEQUENCE MATCHING

Many application domains are characterized by sequences of event intervals, such as sign language [2, 3], medicine [4], geo-informatics [5], cognitive science [6], linguistics [7], and music informatics [139]. For example, in sign language, sentences are constructed by events corresponding to occurrences of various grammatical, syntactic, and gestural expressions. Such expressions have a time duration and they may occur concurrently, hence, sequences of event intervals are formed. Moreover, in medicine [4], patients typically undergo a series of diagnostic tests and treatements that have a time duration and may also occur concurrently.

The main advantage of event-interval sequences over traditional event sequences, which model series of instantaneous events, is that they incorporate the notion of duration in their event representation. Essentially, sequences of event intervals can be encoded as a collection of labeled events accompanied by their start and end time values. An example of a sequence of five labeled temporal intervals is shown in Figure 9.1.

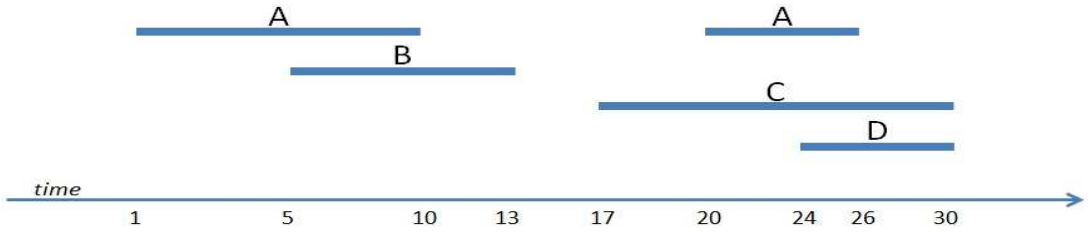


Figure 9.1: Example of a sequence of five event intervals. Four event labels are used in this example: A, B, C, and D. Note that event A occurs twice.

Knowledge discovery tasks have been so far the main focus of many studies on sequences of event intervals. Such tasks include mining patterns and association rules [2, 140, 141, 142], mining semi-interval partial orders [143], and discovering relationships for classification [144]. Surprisingly, similarity searching and matching has received limited attention [145, 146].

Here, we study the problem of *full sequence matching* of sequences of event intervals. Developing robust methods for solving this problem will facilitate a wide range of knowledge discovery tasks (such as classification and clustering) in a wide range of application domains where such sequences occur, like the ones mentioned above. Existing similarity measures on discrete sequences, such as the Levenshtein distance [43], are not directly applicable to sequences of event intervals. As shown by Kostakis et al. [146], mapping a sequence of event intervals to a discrete event sequence, by considering only the start and end points of each event interval and labeling them with the event label that corresponds to that interval, may lead to a large number of false matches.

A recent similarity measure, `Artemis` [146], has been proposed for similarity matching of event-interval sequences. Each sequence is represented by the set of temporal relations between all pairs of event intervals. Given two sequences, and using this representation, `Artemis` finds an optimum matching between their pairs of event intervals by mapping the sequences into a bipartite graph. Then, similarity is inferred using the Hungarian algorithm. One limitation of this method is that it only considers the types of temporal relations in the matching and ignores the actual duration of the events. For instance, consider the three sequences shown in Figure 9.2. Each sequence consists of two event intervals with the same label and the same temporal relation: *overlap*. Hence, `Artemis` would conclude that the three sequences are identical. Nonetheless, one could argue that these sequences differ substantially:

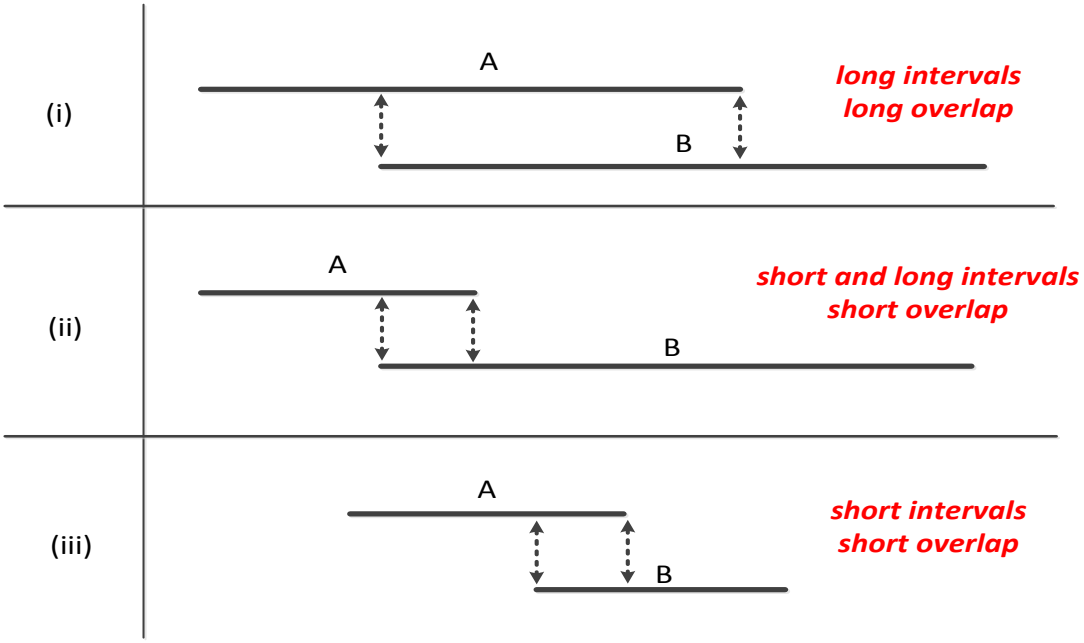


Figure 9.2: Three sequences of two event intervals A and B. In all sequences the temporal relation between the events is the same (they are overlapping), though the time duration of both the intervals and their overlap is different.

the time durations of the intervals vary among the three sequences and the time duration of their overlap is also different.

In this chapter, we address this shortcoming of `Artemis` and present a novel method, `IBSM` (shorthand for Interval-Based Sequence Matching), which performs full sequence matching by: (1) transforming the compared event-interval sequences to a vector-based representation, and (2) computing the Euclidean distance of the new representations of the sequences. The key novelty of `IBSM` is that it explicitly takes into account the time durations of the event intervals in the sequences and implicitly their temporal relations [147].

The main contributions of this chapter include:

- a robust vector-based representation of event-interval sequences that facilitates full sequence matching,

- a novel method, called `IBSM`, for matching event-interval sequences that exploits the vector-based representation by applying bilinear interpolation and computes the Euclidean distance of the resulting sequence representations,
- two techniques based on sampling and alphabet reduction that speed up the runtime and decrease the memory requirements of `IBSM` when performing nearest neighbor search in a database of event-interval sequences, and
- an extensive experimental evaluation of `IBSM` against two state-of-the art measures on eight real datasets taken from different application domains, where `IBSM` is shown to outperform existing state-of-the-art methods in 7 out of 8 datasets, in terms of nearest neighbor classification accuracy, and by up to two orders of magnitude in terms of runtime.

9.1 Related Work

Existing work on temporal interval sequences has so far been focusing merely on frequent pattern and association rule mining. Several approaches [148, 149] consider discovering frequent intervals in databases, where intervals appear sequentially and are not labeled, while others [150] consider temporally annotated sequential patterns where transitions from one event to another have a time duration. A graph-based approach [151] represents each temporal pattern by considering only two types of relations between event intervals (*follow* and *overlap*). In Ale et al. [141], the lifetime of an item is defined as the time between its first and last occurrence and the support is calculated with respect to this interval.

A large variety of Apriori-based techniques [140, 152, 153, 154, 155, 156, 157] for finding temporal patterns, episodes, and association rules on interval-based event sequences have been proposed. BFS-based and DFS-based approaches [2, 3, 158, 159]

apply efficient pruning techniques, thus reducing the inherent exponential complexity of the mining problem, while a non-ambiguous hierarchical representation of interval-based event sequences has been proposed by Patel et al. [160] for frequent pattern mining. In addition, there has been some recent work on mining semi-partial orders of time intervals [143], while an efficient method for mining closed patterns of interval-based events has been proposed [161].

Recent work on margin-closed patterns [143, 162] focuses on significantly reducing the number of reported patterns by favoring longer patterns and suppressing shorter patterns with similar frequencies. A unifying view of temporal concepts and data models has been formulated to enable categorization of existing approaches to unsupervised pattern mining from symbolic temporal data [142]; time point-based methods and interval-based methods as well as univariate and multivariate methods are considered.

A thorough survey of the literature on pattern mining from event-interval sequences is beyond the scope of this chapter as the problem studied here is fundamentally different. To the best of our knowledge, the only existing principled method for comparing event-interval sequences is `Artemis` [146]. A baseline approach, `DTW-based`, which is described in Kostakis et al. [146], is based on an event-interval sequence representation that is vector-based, however, due to its construction, it fails to take into consideration all pair-wise temporal relations [146]. Finally, a matrix-based representation [145] has been proposed for comparing event-interval sequences. Unfortunately this representation is ambiguous, i.e., two different event-interval sequences may have the same matrix representation, and hence it is not considered further.

9.2 Background

Let $\Sigma = \{E_1, \ldots, E_m\}$ be an alphabet of m event labels. An event that occurs over a time interval defines an *event interval* and an ordered multiset of event intervals defines an *event-interval sequence*. Next, we provide a more formal defintion for these two concepts.

Definition 5. (*event interval*) An *event interval* is defined as a triple $S = (E, t_{start}, t_{end})$, where $S.E \in \Sigma$ and $S.t_{start}$, $S.t_{end}$ correspond to the start and end time of S, respectively. In general, $S.t_{start} \leq S.t_{end}$, where the equality holds when the event is instantaneous.

Definition 6. (*e-sequence*) An *event-interval sequence* or *e-sequence* $\mathcal{S}$=$\{S_1, \ldots, S_n\}$ is an ordered multiset of n event intervals. The temporal order of the event intervals in $\mathcal{S}$ is ascending based on their start time and in the case of ties it is descending based on their end time. In case of further ties, alphabetical order is used.

The *size* of an e-sequence $\mathcal{S}$, $|\mathcal{S}|$, is the number of event intervals in the e-sequence, whereas the *length* of $\mathcal{S}$ corresponds to the maximum time point in $\mathcal{S}$, i.e., $length(\mathcal{S}) = S_n.t_{end}$. Recalling the example in Figure 9.1 and following the above definitions, we derive the following e-sequence representation: $\mathcal{S} = \{(A, 1, 10), (B, 5, 13), (C, 17, 30), (A, 20, 26), (D, 24, 30)\}$.

We use Allen's model for temporal logic [163, 164] to define the relations between two event intervals. Given two event intervals A and B, we consider seven temporal relations: *meet*, *match*, *overlap*, $left$-*contain*, *right-contain*, *contain*, and $follow$. These relations are shown in Figure 9.3.

The problem we study here is how to assess the similarity between two e-sequences $\mathcal{S}$ and $\mathcal{T}$. A robust distance measure should take into account several common characteristics of $\mathcal{S}$ and $\mathcal{T}$: (1) common event labels, (2) pairs of event

intervals with the same temporal relation, (3) the time duration of each event interval as well as the duration of each temporal relation.

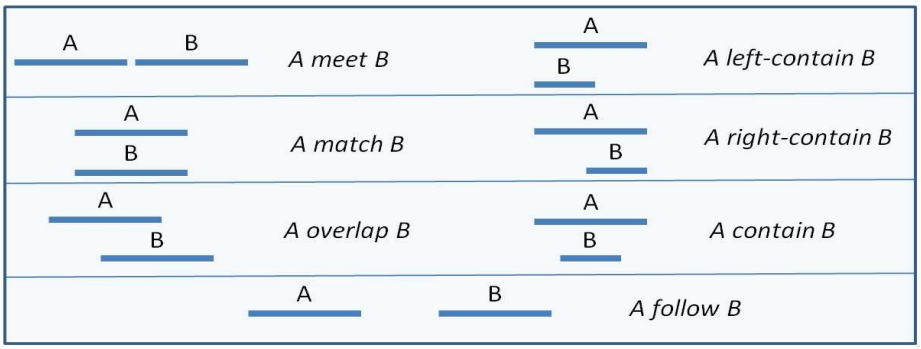


Figure 9.3: Seven temporal relations between two event intervals.

9.3 IBSM

We introduce a novel approach, called `IBSM`, for performing e-sequence matching between two e-sequences. Each e-sequence $\mathcal{S}$ is mapped to its corresponding *resized event table* representation, which is computed in two phases: (a) $\mathcal{S}$ is converted to an *event table* $A_{\mathcal{S}}$, and then (b) $A_{\mathcal{S}}$ is resized to yield the final representation of $\mathcal{S}$. Finally, `IBSM` computes the Euclidean distance of these representations. As shown in Section 9.3.3, `IBSM` performs e-sequence matching in time linear to the length of the e-sequences.

9.3.1 Event table representation

Firstly, we assume that at each point in time all events may appear simultaneously. More than that, even multiple instances of one event are allowed to happen concurrently at a specific time point. Hence, given an e-sequence $\mathcal{S}$ and a time point

t, some events in Σ may be *active* and some may be not. We use an *event vector* to record this information.

Definition 7. (*active event interval*) An event interval $S = (E, t_{start}, t_{end})$ is active at time point t if and only if

$$S.t_{start} \leq t \leq S.t_{end}$$

Definition 8. (*event vector*) Given an e-sequence $\mathcal{S} = \{S_1, \ldots, S_n\}$ and a time point t, the *event vector* $\mathcal{V}_S^t$ of $\mathcal{S}$ at time t is a vector of size $|\Sigma|$, where each value $\mathcal{V}_S^t(i)$ is a non-negative integer that records how many event intervals with label $E_i \in \Sigma$ are active at time t.

Therefore, for each time point $t \in [1, \ldots, length(\mathcal{S})]$, the corresponding event vector $\mathcal{V}_S^t$ is computed and recorded as a column in the *event table* $A_\mathcal{S}$.

Definition 9. (*event table*) Given an e-sequence $\mathcal{S} = \{S_1, \ldots, S_n\}$ its corresponding *event table* $A_\mathcal{S}$ is a matrix of size $|\Sigma| \times length(\mathcal{S})$, where $A_\mathcal{S}(i, j) = \mathcal{V}_S^j(i)$, for $i \in [1, \ldots \Sigma]$ and $j \in [1, \ldots length(\mathcal{S})]$.

In other words, each e-sequence $\mathcal{S}$ is represented by event table $A_\mathcal{S}$, where each cell (i, j) of the table reflects the number of times the i-th event in Σ appears at the j-th time point in $\mathcal{S}$.

9.3.2 Resizing the event table

The next step of our method is to ensure that the tables of the two e-sequences $\mathcal{S}$ and $\mathcal{T}$ under comparison are of the same size. To achieve this, we use a resizing parameter γ. Typically, γ is set to the maximum time point between the two e-sequences, that is

$$\gamma = max\{length(\mathcal{S}), length(\mathcal{T})\}.$$

Note that, in case we are computing pair-wise distances of more than two e-sequences, γ is set to the maximum e-sequence length.

Definition 10. (γ-*resized event table*) Given an event table $A_{\mathcal{S}}$ and a resizing parameter γ, the γ-*rezised event table* $A^{\gamma}_{\mathcal{S}}$ is computed by resizing $A_{\mathcal{S}}$ to γ using *bilinear interpolation*.

Consequently, after this step is executed, $\mathcal{S}$ and $\mathcal{T}$ are represented by their γ-resized event tables $A^{\gamma}_{\mathcal{S}}$ and $A^{\gamma}_{\mathcal{T}}$, respectively, which are of the same size $|\Sigma|$ x γ.

The final step of `IBSM` is to compute the distance of the resulting γ-resized event tables of $\mathcal{S}$ and $\mathcal{T}$. The distance measure used is the Euclidean distance, and it is computed as follows:

$$D(\mathcal{S},\mathcal{T}) = \sqrt{\sum_{i=1}^{|\Sigma|}\sum_{j=1}^{\gamma}(A^{\gamma}_{\mathcal{S}}(i,j) - A^{\gamma}_{\mathcal{T}}(i,j))^2} \tag{9.1}$$

9.3.3 Complexity

The online computational complexity of `IBSM` is linear to the length of the e-sequences. More specifically, given two e-sequences $\mathcal{S}$, $\mathcal{T}$, defined over an event alphabet Σ, and a resizing parameter γ, the online computational time and space complexity of `IBSM` is $O(|\Sigma| \times \gamma)$. Regarding the pre-processing step, the method requires $O(|\Sigma| \times length(\mathcal{S}))$ and $O(|\Sigma| \times length(\mathcal{T}))$ time and space for computing and resizing the event tables for $\mathcal{S}$ and $\mathcal{T}$, respectively.

9.4 Nearest neighbor search using `IBSM`

`IBSM` can be used for nearest neighbor search in an *e-sequence database*, i.e., a collection of e-sequences. Let DB be an e-sequence database. Given a query e-sequence Q we want to find the nearest neighbor of Q in DB. Despite the linear com-

plexity of IBSM, which already achieves over an order of magnitude speedup compared to the state-of-the-art Artemis method, it depends on the lengths of the e-sequences in DB as well as the alphabet size $|\Sigma|$. Thus, we next introduce two techniques for speeding up IBSM for nearest neighbor search, that can achieve significant speedups by reducing the e-sequence lengths as well as the alphabet size.

9.4.1 Speedup by sampling

The first speedup technique that can be applied is to reduce the number of columns of the γ-resized event tables of the e-sequences in DB. Given the γ-resized event table $A^{\gamma}_{\mathcal{S}}$ of each e-sequence $\mathcal{S} \in DB$, we perform uniform sampling on the columns of $A^{\gamma}_{\mathcal{S}}$ with a sampling period equal to δ. This results in including only columns $1, 1+\delta, 1+2\delta, 1+3\delta, \ldots$ from $A^{\gamma}_{\mathcal{S}}$.

More formally, we consider the following set of columns:

$$\{j+1 | j \in [0, \gamma) \text{ and } mod(j, \delta) = 0\}.$$

Sampling results in a reduction on the columns of $A^{\gamma}_{\mathcal{S}}$ expressed by the *sampling rate* $r \in [0, 1]$ given by

$$r = \frac{\lceil \frac{\gamma}{\delta} \rceil}{\gamma}. \tag{9.2}$$

Effectively, the number of columns in $A^{\gamma}_{\mathcal{S}}$ is reduced from γ to $\lceil \frac{\gamma}{\delta} \rceil$, which implies that each γ-resized event table is reduced to $r \times 100\%$ of its original size.

9.4.2 Speedup by alphabet reduction

We pressent a second speedup technique that reduces the number of rows of the γ-resized event tables of the e-sequences in DB. The key idea is to reduce the size of the alphabet Σ. Specifically, given a γ-resized event table $A^{\gamma}_{\mathcal{S}}$ of γ columns, for each

event $E_i \in \Sigma$, the fraction of non-zero occurrences of E_i in row i of the table, denoted as $h(E_i, A^{\gamma}_{\mathcal{S}})$, is given by

$$h(E_i, A^{\gamma}_{\mathcal{S}}) = \frac{1}{\gamma} \sum_{j=1}^{\gamma} I(A^{\gamma}_{\mathcal{S}}(i,j)),$$

where $I(\cdot)$ is an indicator function such that

$$I(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$$

Definition 11. (*event density*) Given a database DB of e-sequences defined over an alphabet Σ, the event density $H(E_i, DB)$ of each $E_i \in \Sigma$ in DB is defined as follows:

$$H(E_i, DB) = \frac{1}{|DB|} \sum_{\mathcal{S} \in DB} h(E_i, A^{\gamma}_{\mathcal{S}}).$$

In other words, the event density of E_i in DB expresses the average fraction of non-zero occurrences of E_i in the γ-resized event tables of the e-sequences in DB.

The alphabet reduction technique we propose here computes the density of each event $E_i \in \Sigma$ in DB and removes E_i from Σ, if

$$H(E_i, DB) < \epsilon \text{ , where } \epsilon \in [0, 1].$$

Practically, ϵ is a threshold placed on the average frequency of event E_i in DB. If E_i appears in less than $\epsilon \times 100\%$ of the columns of the γ-resized event tables in DB, on average, then it is removed from Σ. The intuition behind this technique is that event labels that appear more frequently in active event intervals in DB are those that mostly characterize the e-sequences in DB.

Therefore, the ratio of the reduced alphabet size over the initial size, called the *alphabet reduction rate* $s \in [0, 1]$, is given by

$$s = 1 - \frac{|\{E_i \in \Sigma | H(E_i, DB) < \epsilon\}|}{|\Sigma|}. \tag{9.3}$$

Effectively, the number of rows in $A^{\gamma}_{\mathcal{S}}$ is reduced from $|\Sigma|$ to $s|\Sigma|$, which implies that each γ-resized event table is reduced to $s \times 100\%$ of its original size.

In the description of the two previous techniques we assumed that each technique is applied to the original γ-resized event table. To speed up `IBSM` we apply both techniques and finally compute the distance of the reduced event tables using Equation 9.1. As a result, the overall benefit gained by applying both techniques is a total reduction of each γ-resized event table to $r \times s \times 100\%$ of its original size.

9.5 Experiments

In this section, we present the experimental setup and results evaluating the performance of `IBSM`.

9.5.1 Experimental Setup

In our experiments we used eight real datasets, and compared `IBSM` with two competitor methods in terms of 1-NN classification accuracy and runtime.

9.5.1.1 Datasets

Table 9.1: Datasets Statistics

Dataset	# of	e-sequence size			# of	# of	max e-seq.	interval size			
	e-seq.	min.	max.	average	labels	classes	length	mean	stdev	min	max
ASL-BU	873	3	40	17	216	9	5,901	594	590	3	4,468
ASL-BU2	1,839	4	93	23	254	7	14,968	669	808	3	9,967
Auslan2	200	9	20	12	12	10	30	20	12	1	30
Blocks	210	3	12	6	8	8	123	17	12	1	57
Context	240	47	149	81	54	5	284	69	81	1	284
Hepatitis	498	15	592	108	63	2	7,555	634	1,093	1	7,555
Pioneer	160	36	89	56	92	3	80	36	21	1	80
Skating	530	27	143	44	41	6	6,829	576	672	1	6,829

Our eight real datasets were taken from various application domains. A summary of the statistics for each dataset is shown in Table 9.1. Below, we describe each dataset in more detail:

- ASL-BU [2]. Event labels correspond to grammatical or syntactic forms (e.g., wh-word, wh-question, verb, noun, etc.) as well as facial or gestural expressions (e.g., head tilt right, rapid head shake, eyebrow raise, etc.). An e-sequence is an expression of a sentence using sign language.
- ASL-BU2. This is the newest version of ASL-BU. The structure is the same as that of ASL-BU but this dataset contains a large number of new e-sequences and versions of the previous ones that are improved in terms of annotation, where additional labels have been introduced.
- Auslan [143]. The e-sequences were derived from the Australian Sign Language dataset available in the UCI repository [1]. Each event interval represents a word like *girl* or *right*.
- Blocks [143]. Event labels correspond to visual primitives obtained from videos of a human hand stacking colored blocks and describe which blocks are touched as well as the actions of the hand (e.g., contacts blue or red, attached hand red, etc.). Each e-sequence represents one of eight different scenarios including atomic actions, such as *pickup*, or complete scenarios, such as *assemble*.
- Context [143]. Event labels were derived from categoric and numeric data describing the context of a mobile device carried by humans in different situations. Each e-sequence represents one of five different scenarios such as *street* or *meeting*.

[1] http://www.ics.uci.edu/ mlearn/MLRepository.html

- Hepatitis [165] The dataset contains information about patients who have either Hepatitis B or Hepatitis C. The event intervals represent the results of 63 regular tests. Each e-sequence describes a series of tests taken by a patient.
- Pioneer [143]. This dataset was constructed from the Pioneer-1 dataset available in the UCI repository [2]. Event intervals correspond to the input provided by the robot sensors. Each e-sequence in the dataset describes one of three scenarios: *gripper*, *move*, *turn*.
- Skating [143]. Event intervals describe muscle activity and leg position of 6 professional In-Line Speed Skaters during controlled tests at 7 different speeds on a treadmill. Each e-sequence represents a complete movement cycle.

9.5.1.2 Methods

We compared the following methods:

- `IBSM`: we experimented with and without sampling and alphabet reduction.
- `Artemis` [146]: the state-of-the-art method for similarity mathing of e-sequences.
- `DTW-based` [146]: the baseline approach which `Artemis` has been compared to. Each e-sequence is mapped to a set of vectors. Each vector keeps track of the number of times each event label appears, and is created only for these time points where a change occurs, i.e., when one or more event intervals become active or inactive. The dimensions are combined using the root mean square.

9.5.1.3 Evaluation

We computed the 1-NN *classification accuracy* for each method and dataset. For each dataset, each e-sequence was considered to be a query Q and the distance

[2] http://archive.ics.uci.edu/ml/

from the remaining e-sequences in the dataset was computed. The class label of the 1-NN e-sequence was compared to that of Q. In case of ties a majority scheme was followed.

To be more specific, let DB be the set of e-sequences in a dataset. For each query $Q \in DB$, we computed its distance $D(Q, \mathcal{S})$ against each e-sequence $\mathcal{S} \in DB \setminus Q$. Note that $D(\cdot)$ corresponds to the distance function used by each of the three methods. Let $\mathcal{D}_Q = \{D(Q, \mathcal{S}) | \forall \mathcal{S} \in DB \setminus Q\}$ be the set of distances of Q to $DB \setminus Q$ and $NN_Q = min(\mathcal{D}_Q)$ the distance of the 1-NN of Q in $DB \setminus Q$. If there is only one e-sequence in $DB \setminus Q$ with distance NN_Q from Q then we just compare the class labels of the two e-sequences. If there is more than one e-sequence with that distance, then we consider the union of all classes of these e-sequences and report the majority class as the class of the nearest neighbor.

In the results reported in Section 9.5.2, the classification accuracy for each dataset is defined as the percentage of the total number of e-sequences of the dataset that were correctly classified. We also measured the *runtime*. The methods have been implemented in Matlab on an AMD Opteron 8220 SE processor running at 2.8GHz.

9.5.2 Experimental Results

Next, we present our experimental findings in terms of classification accuracy and runtime.

9.5.2.1 Classification accuracy

We first evaluated the performance of `IBSM` in terms of 1-NN classification accuracy on the eight datasets and compared it to that of `Artemis` and `DTW-based`. For this erxperiment, we did not apply the speedup techniques for `IBSM`, i.e., we set the sampling period $\delta = 1$ and the alphabet reduction threshold $\epsilon = 0$. In Table 9.2

we can see that IBSM is a clear winner in all datasets except for the Pioneer dataset. The results strongly suggest that when comparing e-sequences it is essential to take into account the interval durations along with the relation types (which is what IBSM does), since the 1-NN classification accuracy can be substantially improved.

Next, we studied the effect of sampling and alphabet reduction on the accuracy of IBSM when performing nearest neighbor search. We first experimented on different sampling rates r keeping the alphabet size fixed to $|\Sigma|$. In Figures 9.4 and 9.5 we show the performance of IBSM in terms of 1-NN classification accuracy for different sampling rates, compared to Artemis and DTW-based. We observed that for all datasets a sampling rate of $r = 10\%$ suffices to maintain the original 1-NN classification accuracy (i.e., without sampling). Note that for ASL-BU, ASL-BU2, Hepatitis, and Skating the rate was $\approx 0.9\%$.

In addition, we experimented on different alphabet reduction rates while maintaining the sampling rate constant. Using the result of the previous experiment, we fixed the sampling rate to $r = 10\%$, and varied the alphabet reduction rate s. The results are shown in Figures 9.6 and 9.7. Observe that the effect of s on the 1-NN classification accuracy varies per dataset. Specifically, ASL-BU, Blocks, and Skating can tolerate a reduction down to $s = 40\%$ without significant drop in the accuracy, while for Hepatitis and Pioneer the original accuracy is mantaintaned until s drops down to 30%. The results are quite worse for Context, where $s = 50\%$, and Auslan2, where $s = 70\%$. Finally, ASL-BU2 shows the best performance as it can tolerate an alphabet reduction down to $s = 10\%$ without significant loss in terms of 1-NN classification accuracy.

We conclude that in the majority of the datasets applying a sampling rate r of 10% and reducing the alphabet size by 50%, hence reducing the computational cost

Table 9.2: 1-NN classification accuracy. For IBSM neither sampling nor aplhabet reduction have been applied.

Dataset	IBSM	Artemis	DTW
ASL-BU	**90.1**	79.56	43.58
ASL-BU2	**82.2**	80.53	77.25
Auslan2	**39.5**	28.5	22
Blocks	**100**	99	87
Context	**97.08**	90	89
Hepatitis	**77.91**	72.09	74.03
Pioneer	93.75	**97.5**	93
Skating	**97.74**	84	77

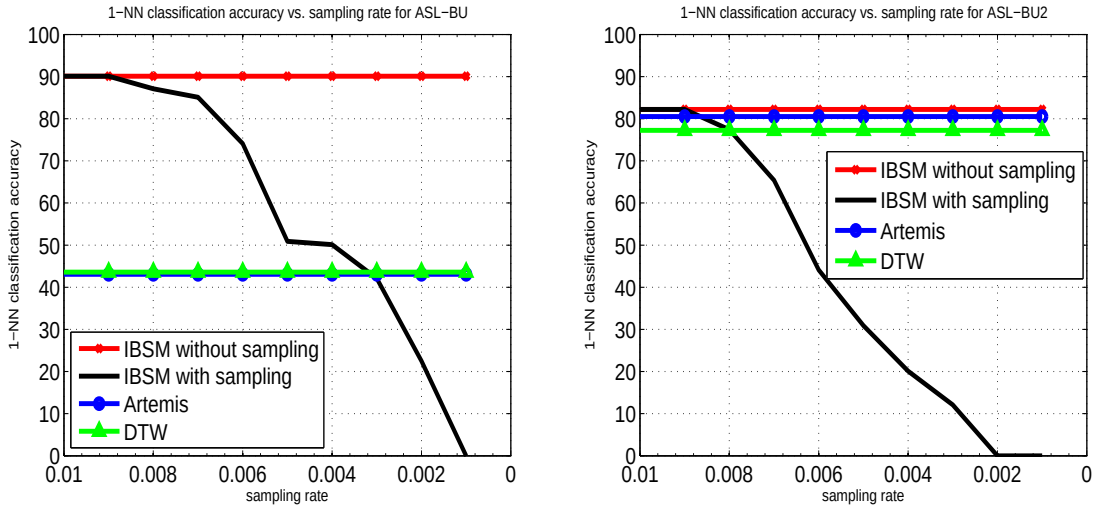
by a factor of 95%, can still maintain a 1-NN classification accuracy that is higher than that of the competitor methods for 7 out of 8 datasets.

Table 9.3: Runtime in seconds. We show the average total runtime (including pre-processing and matching) for comparing a pair of e-sequences. For IBSM neither sampling nor alphabet reduction have been applied.

Dataset	IBSM	Artemis	DTW
ASL-BU	**81.34**	1,915.97	190.89
ASL-BU2	**659.73**	14,018.08	2,140.47
Auslan2	**0.94**	9.64	2.73
Blocks	**0.79**	22.20	1.56
Context	**5.53**	121.66	9.56
Hepatitis	**27.96**	537.16	58.23
Pioneer	**8.76**	98.41	25.48
Skating	**14.96**	259.63	39.90

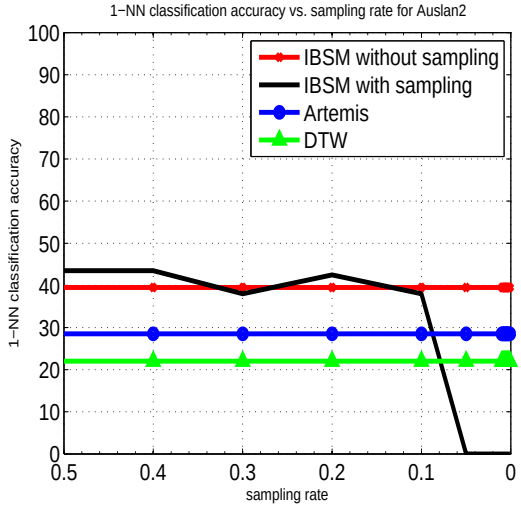
9.5.2.2 Runtime

Finally, we benchmarked the methods in terms of runtime. The results are shown in Table 9.3. Note that for IBSM neither sampling nor alphabet reduction have been applied. For each dataset, we show the average total runtime (including pre-

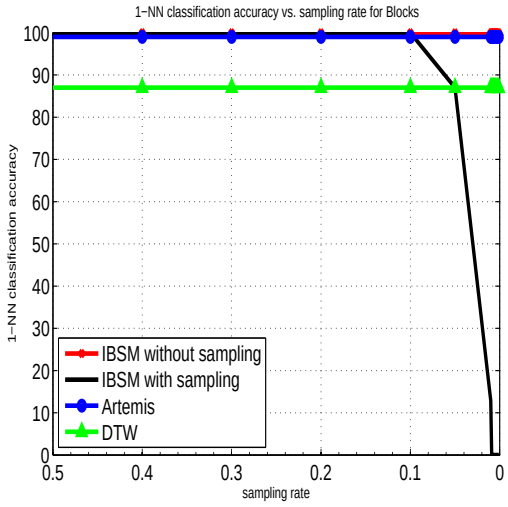


(a) *ASL-BU*

(b) *ASL-BU2*



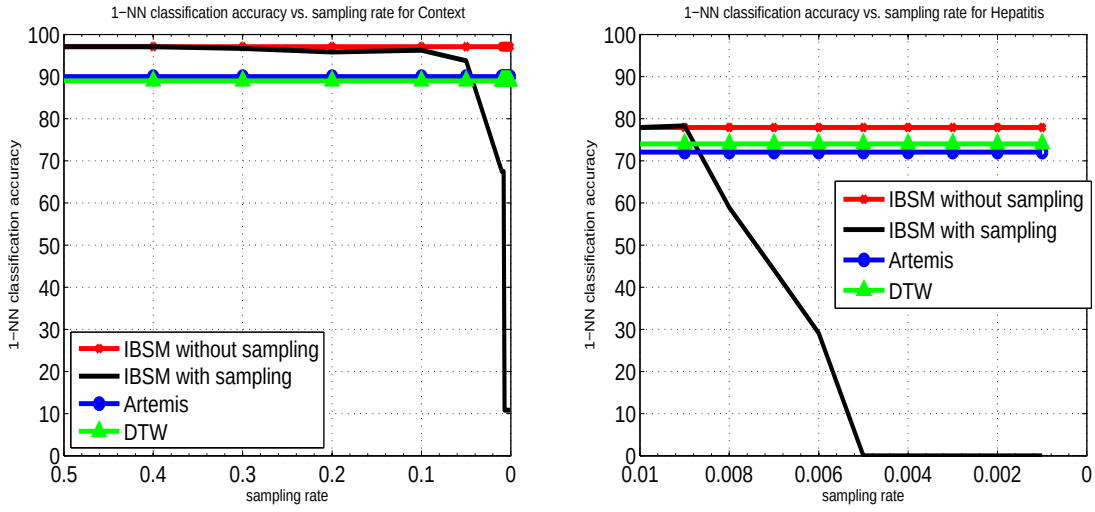
(c) *Auslan2*



(d) *Blocks*

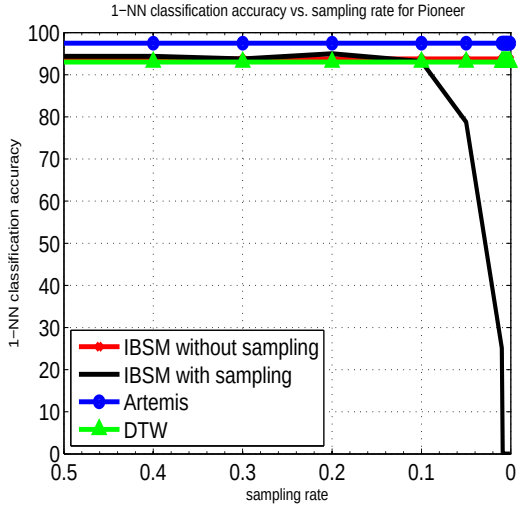
Figure 9.4: Comparison of `IBSM`, `Artemis`, and `DTW-based` for different sampling rates r for the first four datasets. No alphabet reduction was applied ($\epsilon = 0$). The flat lines are used to indicate the 1-NN accuracy of the two competitor methods and `IBSM` without sampling.

processing and matching) for comparing a pair of e-sequences. It turns out that `IBSM` is a clear winner in all cases achieving up to two orders of magnitude speedup against

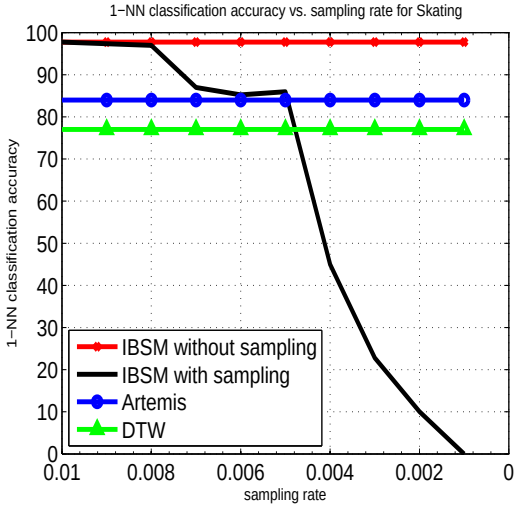


(a) *Context*

(b) *Hepatitis*



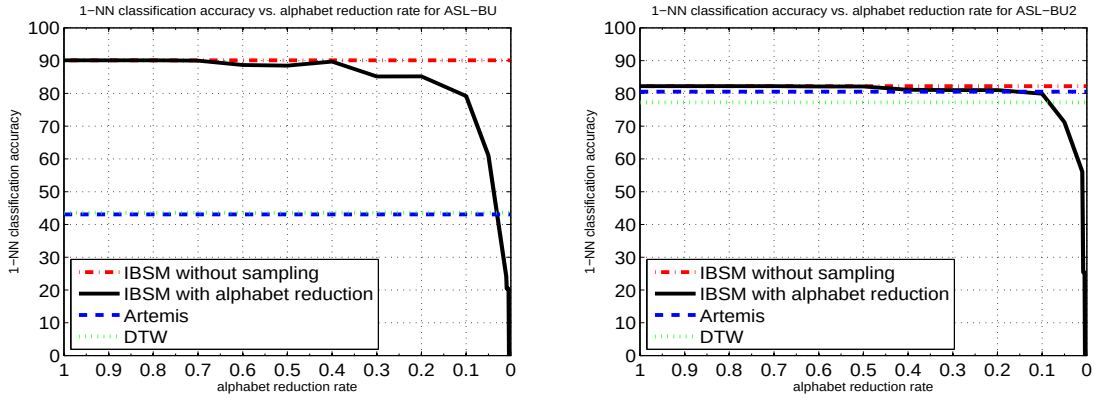
(c) *Pioneer*



(d) *Skating*

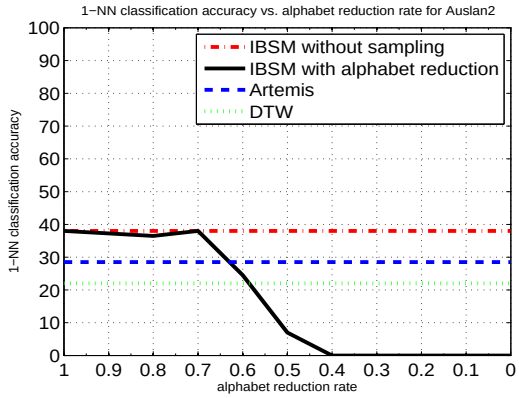
Figure 9.5: Comparison of `IBSM`, `Artemis`, and `DTW-based` for different sampling rates r for the last four datasets. No alphabet reduction was applied ($\epsilon = 0$). The flat lines are used to indicate the 1-NN accuracy of the two competitor methods and `IBSM` without sampling.

`Artemis`. This speedup becomes significantly higher (by at least another order of magnitude) when applying sampling and alphabet reduction.

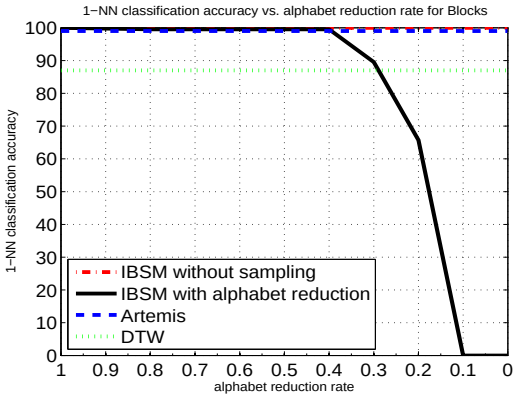


(a) *ASL-BU*

(b) *ASL-BU2*



(c) *Auslan2*

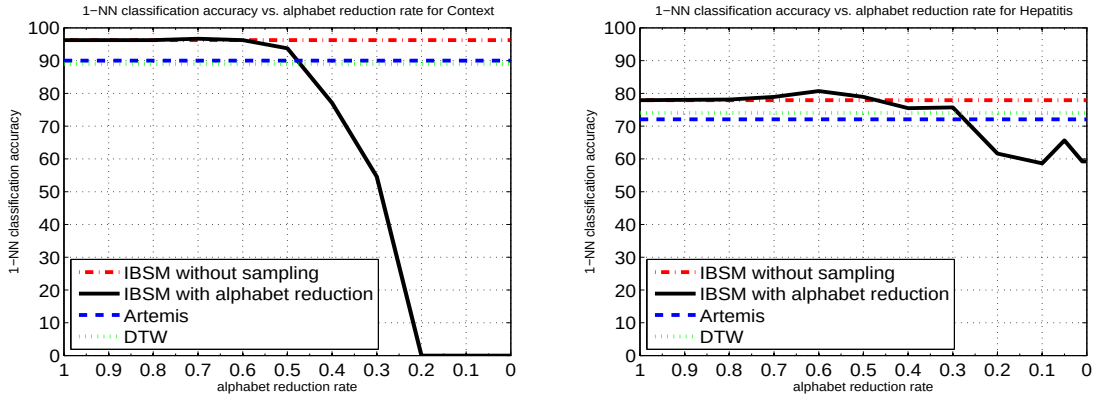


(d) *Blocks*

Figure 9.6: Comparison of `IBSM`, `Artemis`, and `DTW-based` for different alphabet reduction rates s for the first four datasets. Note that the sampling rate for `IBSM` was fixed to $r = 10\%$. The flat lines are used to indicate the 1-NN accuracy of the two competitor methods and `IBSM` without sampling.

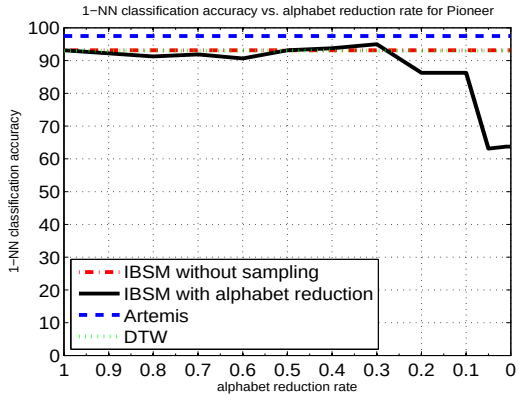
9.6 Conclusions

We have proposed a novel method for full matching of sequences of interval-based events. The novelty of the method against existing approaches is the fact that it considers both temporal relations and duration of the event intervals in the e-sequences. The method converts the original sequences to an event table representation and then computes a Euclidean-based distance between the event tables.

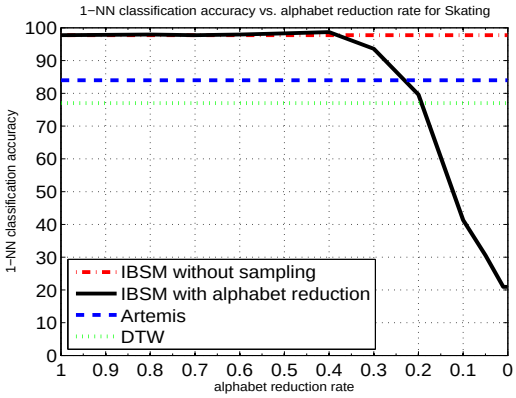


(a) *Context*

(b) *Hepatitis*



(c) *Pioneer*



(d) *Skating*

Figure 9.7: Comparison of `IBSM`, `Artemis`, and `DTW-based` for different alphabet reduction rates s for the last four datasets. Note that the sampling rate for `IBSM` was fixed to $r = 10\%$. The flat lines are used to indicate the 1-NN accuracy of the two competitor methods and `IBSM` without sampling.

Additionally, we have presented two techniques for speeding up `IBSM` when used for nearest neighbor search in large e-sequence databases. We provided an extensive experimental evaluation of `IBSM` against two state-of-the-art methods on eight real datasets. The performance of our method in terms of 1-NN classification accuracy and runtime is significantly better than the two competitors. Directions for future work include the exploration of additional speedup techniques as well as the theoretical analysis of the properties of the proposed method.

CHAPTER 10

DISCUSSION AND CONCLUSIONS

This thesis described methods for similarity search in large and noisy sequence databases. The different uses that such search can have are illustrated with applications such as music retrieval, time series analysis, and sign language recognition.

The fundamental problems in similarity search are accuracy and efficiency. Accuracy may refer to how well results agree with human judgment, or with how well an efficient approximation preserves the information of the slower method that it approximates. The thesis has proposed five methods that can be applied on time series and event-interval sequences and improve retrieval accuracy, by producing results that are more in line with human expectations. For music retrieval, we have proposed SMBGT, which performs subsequence matching for time series and provides much better results for the very challenging and interesting Query-By-Humming application domain. Motivated by SMBGT, the "Hum-a-song" system has also been proposed for the QBH application, which is an open source system and significantly contributes to this domain. The SMBGT measure, however, can be applied to any other time series domain, and even for whole sequence matching. In addition, we have proposed three techniques for searching time series databases. Two of them are able to select for each query the best distance measure out of a pool of measures via statistical significance testing, so as to perform nearest neighbor classification on time series. The third one exploits the Hidden Markov Models, which are capable of modeling classes of time series, and has been applied to searching time series databases. The results of the aforementioned three measures on several public datasets demonstrate that

they provide better accuracy compared to several baseline methods in the majority of datasets. The fifth method proposed, named IBSM, targets event-interval sequences, and has been shown to outperform the state-of-the-art methods in both classification accuracy and runtime on most datasets tested, including sign language.

With respect to improving retrieval efficiency, the thesis has proposed two methods that significantly improve efficiency in two target domains. The first method, ISMBGT, focuses on the topic of speeding up retrieval on the QBH domain. Thus, it is built on top of the SMBGT method, which is very flexible and can take into account several intrinsic properties of this noisy domain, and works in a filter-and-refine way by exploiting the idea of embeddings. The second method, MTSI, takes advantage of the novel model-based representation of classes of time series, and operates again in a filter-and-refine manner. The performance evaluation of both methods on searching and classifying time series has proved how meaningful and accurate they are.

While the proposed methods have extended the state-of-the-art in their respective target domains, there are still important challenges remaining. First, since the complexity of the query-selection methods proposed in the thesis can be prohibitive for very large time series databases, especially when many distance measures compose the pool of measures, their online step has to be made faster. Second, event-interval sequences have received limited attention from the database community. Thus, extending our approach, or even proposing a new one, for subsequence matching in event-interval sequences would be particularly useful. Finally, although the model-based approaches that have been proposed produce very accurate results, there is still room for improvement by making the Forward algorithm perform faster, and by training the Hidden Markov Models with different parameters for each class they represent. These challenges, among others, are of particular interest, and hence will be part of future work.

REFERENCES

[1] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[2] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Mining frequent arrangements of temporal intervals," *Knowledge and Information Systems (KAIS)*, pp. 133–171, 2009.

[3] ——, "Discovering frequent arrangements of temporal intervals," in *International Conference on Data Mining (ICDM)*, 2005.

[4] R. Kosara and S. Miksch, "Visualizing complex notions of time," *Studies in Health Technology and Informatics*, pp. 211–215, 2001.

[5] N. Pissinou, I. Radev, and K. Makki, "Spatio-temporal modeling in video and multimedia geographic information systems," *GeoInformatica*, vol. 5, no. 4, pp. 375–409, 2001.

[6] B. Berendt, "Explaining preferred mental models in Allen inferences with a metrical model of imagery," in *Conference of the Cognitive Science Society (CogSci)*, 1996, pp. 489–494.

[7] B. Bergen and N. Chang, "Embodied construction grammar in simulation-based language understanding," *Construction grammars: Cognitive grounding and theoretical extensions*, pp. 147–190, 2005.

[8] A. Kotsifakos, P. Papapetrou, J. Hollmén, D. Gunopulos, and V. Athitsos, "A survey of query-by-humming similarity methods," in *PErvasive Technologies Related to Assistive Environments (PETRA)*, 2012, pp. 5:1–5:4.

[9] M. Clausen, R. Engelbrecht, D. Meyer, and J. Schmitz, "Proms: A web-based tool for searching in polyphonic music," in *International Society for Music Information Retrieval (ISMIR)*, 2000.

[10] A. Lubiw and L. Tanur, "Pattern matching in polyphonic music as a weighted geometric translation problem," in *International Society for Music Information Retrieval (ISMIR)*, 2004, pp. 289–296.

[11] R. Typke, P. Giannopoulos, R. Veltkamp, F. Wiering, and R. Van Oostrum, "Using transportation distances for measuring melodic similarity," in *International Society for Music Information Retrieval (ISMIR)*, 2003, pp. 107–114.

[12] E. Ukkonen, K. Lemström, and V. Mäkinen, "Geometric algorithms for transposition invariant content-based music retrieval," in *International Society for Music Information Retrieval (ISMIR)*, 2003, pp. 193–199.

[13] G. Wiggins, K. Lemström, and D. Meredith, "SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval," in *International Society for Music Information Retrieval (ISMIR)*, 2002, pp. 13–17.

[14] C. Yang, "Efficient acoustic index for music retrieval with various degrees of similarity," in *International Conference on Multimedia*, 2002, pp. 584–591.

[15] J. Downie, "The musifind music information retrieval project, phase iii: evaluation of indexing options," in *Canadian Association for Information Science (CAIS)*, 1995, pp. 135–146.

[16] A. Ghias, J. Logan, D. Chamberlin, and B. Smith, "Query by humming: Musical information retrieval in an audio database," in *International Conference on Multimedia*, 1995, pp. 231–236.

[17] R. McNab, L. Smith, I. Witten, C. Henderson, and S. Cunningham, "Towards the digital music library: Tune retrieval from acoustic input," in *International Conference on Digital Libraries (ICDL)*, 1996, pp. 11–18.

[18] A. Uitdenbogerd and J. Zobel, "Manipulation of music for melody matching," in *International Conference on Multimedia*, 1998, pp. 235–240.

[19] M. Mongeau and D. Sankoff, "Comparison of musical sequences," *Computers and the Humanities*, vol. 24, no. 3, pp. 161–175, 1990.

[20] A. Uitdenbogerd and J. Zobel, "Melodic matching techniques for large music databases," in *International Conference on Multimedia (Part 1)*, 1999, pp. 57–66.

[21] B. Pardo and W. Birmingham, "Encoding timing information for musical query matching," in *International Society on Music Information Retrieval (ISMIR)*, 2002, pp. 267–268.

[22] T. Kageyama, K. Mochizuki, and Y. Takashima, "Melody retrieval with humming," in *International Computer Music Conference (ICMC)*, 1993, pp. 349–349.

[23] L. Ye and E. Keogh, "Time series shapelets: a novel technique that allows accurate, interpretable and fast classification," *Data Mining and Knowledge Discovery (DAMI)*, vol. 22, no. 1-2, pp. 149–182, 2011.

[24] Z. Xing, J. Pei, P. S. Yu, and K. Wang, "Extracting interpretable features for early classification on time series," in *SIAM International Conference on Data Mining (SDM)*, 2011, pp. 247–258.

[25] A. Mueen, E. Keogh, and N. Young, "Logical-shapelets: an expressive primitive for time series classification," in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2011, pp. 1154–1162.

[26] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, ser. KDD '12, 2012, pp. 289–297.

[27] B. Hu, Y. Chen, and E. Keogh, "Time series classification under more realistic assumptions," in *SIAM International Conference on Data Mining (SDM)*, 2012, pp. 578–586.

[28] Y. Chen, M. A. Nascimento, B. Chin, O. Anthony, and K. H. Tung, "Spade: On shape-based pattern detection in streaming time series," in *International Conference on Data Engineering (ICDE)*, 2007, pp. 786–795.

[29] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in *Foundations of Data Organization and Algorithms (FODO)*. Springer Verlag, 1993, pp. 69–84.

[30] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Special Interest Group on Management of Data (SIGMOD) workshop on Research issues in data mining and knowledge discovery (DMKD)*, 2003, pp. 2–11.

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

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

[33] J. B. Kruskall and M. Liberman, "The symmetric time warping algorithm: From continuous to discrete," in *Time Warps*. Addison-Wesley, 1983.

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

[35] I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl, "Anticipatory dtw for efficient similarity search in time series databases," *Proceedings of Very Large Data Bases (PVLDB)*, vol. 2, no. 1, pp. 826–837, 2009.

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

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

[38] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[39] L. Chen and M. T. Özsu, "Robust and fast similarity search for moving object trajectories," in *Special Interest Group on Management of Data (SIGMOD)*, 2005, pp. 491–502.

[40] L. Chen and R. Ng, "On the marriage of l_p-norms and edit distance," in *Very Large Data Bases (VLDB)*, 2004, pp. 792–803.

[41] E. Keogh, "Exact indexing of dynamic time warping," in *Very Large Data Bases (VLDB)*, 2002, pp. 406–417.

[42] A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," *Transactions on Knowledge and Data Engineering (TKDE)*, 2012.

[43] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics*, vol. 10, no. 8, pp. 707–710, 1966.

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

[45] N. Adams, M. Bartsch, J. Shifrin, and G. Wakefield, "Time series alignment for music information retrieval," in *International Society on Music Information Retrieval (ISMIR)*, 2004, pp. 303–311.

[46] R. Dannenberg and N. Hu, "Understanding search performance in query-by-humming systems," in *International Society on Music Information Retrieval (ISMIR)*, 2004, pp. 232–237.

[47] D. Mazzoni and R. Dannenberg, "Melody matching directly from audio," in *International Society on Music Information Retrieval (ISMIR)*, 2001, pp. 17–18.

[48] D. Maier, "The complexity of some problems on subsequences and supersequences," *Journal of the ACM (JACM)*, vol. 25, no. 2, pp. 322–336, 1978.

[49] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *String Processing and Information Retrieval (SPIRE)*, 2000, pp. 39–48.

[50] B. Bollobás, G. Das, D. Gunopulos, and H. Mannila, "Time-series similarity problems and well-separated geometric sets," in *Symposium on Computational Geometry (SOCG)*, 1997, pp. 454–456.

[51] C. Iliopoulos and M. Kurokawa, "String matching with gaps for musical melodic recognition," in *Prague Stringology Club (PSC)*, 2002, pp. 55–64.

[52] M. Crochemore, C. Iliopoulos, C. Makris, W. Rytter, A. Tsakalidis, and K. Tsichlas, "Approximate string matching with gaps," *Nordic Journal of Computing*, vol. 9, no. 1, pp. 54–65, 2002.

[53] T. Crawford, C. Iliopoulos, and R. Raman, "String matching techniques for musical similarity and melodic recognition," *Computing in Musicology*, vol. 11, pp. 73–100, 1998.

[54] K. Lemström and E. Ukkonen, "Including interval encoding into edit distance based music comparison and retrieval," in *Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science (AISB)*, 2000, pp. 53–60.

[55] S. Deorowicz, "Speeding up transposition-invariant string matching," *Information Processing Letters*, vol. 100, no. 1, pp. 14–20, 2006.

[56] V. Makinen, G. Navarro, and E. Ukkonen, "Algorithms for transposition invariant string matching," *Lecture notes in computer science*, pp. 191–202, 2003.

[57] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz, "Similarity search on time series based on threshold queries," in *International Conference on Extending Database Technology (EDBT)*. Springer, 2006, pp. 276–294.

[58] E. Frentzos, K. Gratsias, and Y. Theodoridis, "Index-based most similar trajectory search," in *International Conference on Data Engineering (ICDE)*, 2007, pp. 816–825.

[59] G. E. Batista, E. J. Keogh, O. M. Tataw, and V. M. de Souza, "CID: an efficient complexity-invariant distance for time series," *Data Mining and Knowledge Discovery*, pp. 1–36, 2013.

[60] T. Han, S.-K. Ko, and J. Kang, "Efficient subsequence matching using the longest common subsequence with a dual match index," in *Machine Learning and Data Mining in Pattern Recognition (MLDM)*, 2007, pp. 585–600.

[61] Y. Zhu and D. Shasha, "Warping indexes with envelope transforms for query by humming." in *Special Interest Group on Management of Data (SIGMOD)*, 2003, pp. 181–192.

[62] N. Hu, R. Dannenberg, and A. Lewis, "A probabilistic model of melodic similarity," in *Internationl Computer Music Conference (ICMC)*, 2002, pp. 509–515.

[63] J. Jang and M. Gao, "A query-by-singing system based on dynamic programming," in *International Workshop on Intelligent Systems Resolutions*, 2000, pp. 85–89.

[64] K. Lemström and S. Perttu, "Semex-an efficient music retrieval prototype," in *International Society on Music Information Retrieval (ISMIR)*, 2000, pp. 23–25.

[65] S. Pauws, "Cubyhum: A fully operational query by humming system," in *International Society on Music Information Retrieval (ISMIR)*, 2002, pp. 187–196.

[66] E. Unal, E. Chew, P. Georgiou, and S. Narayanan, "Challenging uncertainty in query by humming systems: a fingerprinting approach," *Transactions on Audio Speech and Language Processing (ASSP)*, vol. 16, no. 2, pp. 359–371, 2008.

[67] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," in *International Conference on Data Engineering (ICDE)*, 2007, pp. 1046–1055.

[68] M. Zhou and M. Wong, "Efficient online subsequence searching in data streams under dynamic time warping distance," in *International Conference on Data Engineering (ICDE)*, 2008, pp. 686–695.

[69] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[70] Z. Ghahramani and M. I. Jordan, "Factorial hidden markov models," *Journal of Machine Learning Research (JMLR)*, vol. 29, pp. 245–275, 1997.

[71] C. Meek and W. Birmingham, "A comprehensive trainable error model for sung music queries," *Journal of Artificial Intelligence Research (JAIR)*, vol. 22, no. 1, pp. 57–91, 2004.

[72] B. Pardo, J. Shifrin, and W. Birmingham, "Name that tune: A pilot study in finding a melody from a sung query," *Journal of the American Society for*

Information Science and Technology (JASIST), vol. 55, no. 4, pp. 283–300, 2004.

[73] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[74] J. Shifrin, B. Pardo, C. Meek, and W. Birmingham, "HMM-based musical query retrieval," in *Joint Conference on Digital Libraries (JCDL)*, 2002, pp. 295–300.

[75] H. Shih, S. Narayanan, and C. Kuo, "An HMM-based approach to humming transcription," in *International Conference on Multimedia and Expo (ICME)*, 2002.

[76] H.-H. Shih, S. S. Narayanan, and C.-C. Kuo, "A statistical multidimensional humming transcription using phone level hidden markov models for query by humming systems," in *International Conference on Multimedia and Expo (ICME)*, vol. 1. IEEE, 2003, pp. I–61.

[77] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," *Theoretical Computer Science (TCS)*, vol. 92, no. 1, pp. 191–211, 1992.

[78] R. Dannenberg, W. Birmingham, B. Pardo, N. Hu, C. Meek, and G. Tzanetakis, "A comparative evaluation of search techniques for query-by-humming using the MUSART testbed," *Journal of the American Society for Information Science and Technology (JASIST)*, vol. 58, no. 5, pp. 687–701, 2007.

[79] A. Kotsifakos, P. Papapetrou, J. Hollmén, and D. Gunopulos, "A Subsequence Matching with Gaps-Range-Tolerances Framework: A Query-By-Humming Application," *Proceedings of Very Large Data Bases (PVLDB)*, vol. 4, no. 11, pp. 761–771, 2011.

[80] C. Meek, J. M. Patel, and S. Kasetty, "OASIS: An online and accurate technique for local-alignment searches on biological sequences," in *Very Large Data Bases (VLDB)*, 2003, pp. 910–921.

[81] E. Keogh, J. Lin, and A. W.-C. Fu, "Hot sax: Efficiently finding the most unusual time series subsequence," in *International Conference on Data Mining (ICDM)*, 2005, pp. 226–233.

[82] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and Knowledge Discovery (DAMI)*, vol. 15, no. 2, pp. 107–144, 2007.

[83] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron, "Q-gram based database searching using a suffix array (quasar)," in *International Conference on Computational Molecular Biology (RECOMB)*, 1999, pp. 77–83.

[84] G. Navarro and R. Baeza-yates, "A new indexing method for approximate string matching," in *Symposium on Combinatorial Pattern Matching (CPM)*, 1999, pp. 163–185.

[85] P. Papapetrou, V. Athitsos, G. Kollios, and D. Gunopulos, "Reference-based alignment of large sequence databases," in *Very Large Data Bases (VLDB)*, 2009.

[86] S. Park, W. W. Chu, J. Yoon, and J. Won, "Similarity search of time-warped subsequences via a suffix tree." *Journal of Information Systems (JIS)*, vol. 28, no. 7, 2003.

[87] S. Park, S. Kim, and W. W. Chu, "Segment-based approach for subsequence searches in sequence databases." in *Symposium on Applied Computing (SAC)*, 2001, pp. 248–252.

[88] E. Keogh and M. Pazzani, "Scaling up dynamic time warping for data mining applications," in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2000.

[89] Y. Shou, N. Mamoulis, and D. Cheung, "Fast and exact warping of time series using adaptive segmental approximations," *Journal of Machine Learning (JMLR)*, vol. 58, no. 2-3, pp. 231–267, 2005.

[90] W.-S. Han, J. Lee, Y.-S. Moon, and H. Jiang, "Ranked subsequence matching in time-series databases," in *Very Large Data Bases (VLDB)*, 2007, pp. 423–434.

[91] A. W.-C. Fu, E. Keogh, L. Y. H. Lau, C. Ratanamahatana, and R. C.-W. Wong, "Scaling and time warping in time series querying," *The Very Large DataBases (VLDB) Journal*, vol. 17, no. 4, pp. 899–921, 2008.

[92] G. Hjaltason and H. Samet, "Properties of embedding methods for similarity searching in metric spaces," *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 25, no. 5, pp. 530–549, 2003.

[93] C. Faloutsos and K. I. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Special Interest Group on Management of Data (SIGMOD)*, 1995, pp. 163–174.

[94] X. Wang, J. T. L. Wang, K. I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang, "An index structure for data mining and clustering," *Knowledge and Information Systems (KAIS)*, vol. 2, no. 2, pp. 161–184, 2000.

[95] G. Hristescu and M. Farach-Colton, "Cluster-preserving embedding of proteins," CS Department, Rutgers University, Tech. Rep. 99-50, 1999.

[96] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios, "BoostMap: A method for efficient approximate similarity rankings," in *Computer Vision and Pattern Recognition (CVPR)*, 2004, pp. 268–275.

[97] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff, "Query-sensitive embeddings," in *Special Interest Group on Management of Data (SIGMOD)*, 2005, pp. 706–717.

[98] P. Papapetrou, V. Athitsos, M. Potamias, G. Kollios, and D. Gunopulos, "Embedding-based subsequence matching in time-series databases," *Transactions on Database Systems (TODS)*, vol. 36, no. 3, p. 17, 2011.

[99] S. Levinson, L. Rabiner, and M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *The Bell System Technical Journal*, vol. 62, no. 4, pp. 1035–1074, 1983.

[100] R. Dannenberg, W. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo, "The Musart Testbed for Query-by-Humming Evaluation," *Computer Music Journal (CMJ)*, vol. 28, no. 2, pp. 34–48, 2004.

[101] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," in *Data mining in Time Series Databases, published by World Scientific*. Publishing Company, 1993, pp. 1–22.

[102] A. Kotsifakos, P. Papapetrou, J. Hollmén, D. Gunopulos, V. Athitsos, and G. Kollios, "Hum-a-song: a subsequence matching with gaps-range-tolerances query-by-humming system," *Proceedings of Very Large Data Bases (PVLDB)*, vol. 5, no. 12, pp. 1930–1933, 2012.

[103] J. Lee and J. Downie, "Survey of music information needs, uses, and seeking behaviours: Preliminary findings," in *International Conference on Music Information Retrieval (ICMIR)*, 2004, pp. 441–446.

[104] N. Scaringella, G. Zoia, and D. Mlynek, "Automatic genre classification of music content: a survey," *Signal Processing Magazine*, vol. 23, no. 2, pp. 133–141, 2006.

[105] F. Fabbri, "A theory of musical genres: Two applications," *Popular Music Perspectives*, vol. 1, pp. 52–81, 1982.

[106] J. Aucouturier and F. Pachet, "Representing musical genre: A state of the art," *Journal of New Music Research*, vol. 32, no. 1, pp. 83–93, 2003.

[107] C. McKay and I. Fujinaga, "Automatic music classification and the importance of instrument identification," in *Conference on Interdisciplinary Musicology (CIM)*, 2005.

[108] ——, "Musical genre classification: Is it worth pursuing and how can it be improved?" in *International Society on Music Information Retrieval (ISMIR)*, 2006, pp. 101–106.

[109] S. Lippens, J. Martens, and T. De Mulder, "A comparison of human and automatic musical genre classification," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 4, 2004, pp. 233–236.

[110] F. Pachet, D. Cazaly *et al.*, "A taxonomy of musical genres," in *Content-Based Multimedia Information Access (RIAO)*, 2000, pp. 1238–1245.

[111] D. Perrot and R. Gjerdigen, "Scanning the dial: An exploration of factors in the identification of musical style," in *Society for Music Perception and Cognition (SMPC)*, 1999, p. 88.

[112] A. Berenzweig, D. Ellis, and S. Lawrence, "Anchor space for classification and similarity measurement of music," in *International Conference on Multimedia and Expo (ICME)*, vol. 1, 2003, pp. 1–29.

[113] Z. Cataltepe, Y. Yaslan, and A. Sonmez, "Music genre classification using midi and audio features," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, pp. 275–279, 2007.

[114] T. Lidy and A. Rauber, "Evaluation of feature extractors and psycho-acoustic transformations for music genre classification," in *International Society on Music Information Retrieval (ISMIR)*, 2005, pp. 34–41.

[115] M. Mandel and D. Ellis, "Song-level features and support vector machines for music classification," in *International Society on Music Information Retrieval (ISMIR)*, 2005, pp. 594–599.

[116] H. Soltau, T. Schultz, M. Westphal, and A. Waibel, "Recognition of music types," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2, 1998, pp. 1137–1140.

[117] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kégl, "Aggregate features and adaboost for music classification," *Journal of Machine Learning (JMLR)*, vol. 65, no. 2, pp. 473–484, 2006.

[118] R. Cilibrasi, P. Vitányi, and R. Wolf, "Algorithmic clustering of music based on string compression," *Computer Music Journal (CMJ)*, vol. 28, no. 4, pp. 49–67, 2004.

[119] R. Dannenberg, B. Thom, and D. Watson, "A machine learning approach to musical style recognition," in *Internationl Computer Music Conference (ICMC)*, 1997, pp. 344–347.

[120] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[121] Z. Fu, G. Lu, K. Ting, and D. Zhang, "A survey of audio-based music classification and annotation," *Transactions on Multimedia*, vol. 13, no. 2, pp. 303–319, 2011.

[122] A. Kotsifakos, E. E. Kotsifakos, P. Papapetrou, and V. Athitsos, "Genre classification of symbolic music with SMBGT," in *PErvasive Technologies Related to Assistive Environments (PETRA)*, 2013.

[123] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

[124] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. Ratanamahatana, "The UCR time series classification/clustering homepage: www.cs.ucr.edu/~eamonn/time_series_data/," 2011.

[125] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 18, no. 6, pp. 607–616, 1996.

[126] C. Domeniconi, J. Peng, and D. Gunopulos, "Locally adaptive metric nearest-neighbor classification," *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 24, no. 9, pp. 1281–1285, 2002.

[127] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff, "Query-sensitive embeddings," *Transactions on Database Systems (TODS)*, vol. 32, no. 2, Jun. 2007.

[128] D. Cox, *Principles of Statistical Inference*. Cambridge University Press, 2006.

[129] V. Athitsos, P. Papapetrou, M. Potamias, G. Kollios, and D. Gunopulos, "Approximate embedding-based subsequence matching of time series," in *Special Interest Group on Management of Data (SIGMOD)*, 2008, pp. 365–378.

[130] E. Keogh, "Exact indexing of dynamic time warping," in *Very Large Data Bases (VLDB)*, 2002, pp. 406–417.

[131] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[132] A. Pikrakis, S. Theodoridis, and D. Kamarotos, "Classification of musical patterns using variable duration hidden Markov models," *Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1795–1807, 2006.

[133] A. Kotsifakos, V. Athitsos, P. Papapetrou, J. Hollmén, and D. Gunopulos, "Model-based search in large time series databases," in *PErvasive Technologies Related to Assistive Environments (PETRA)*, 2011.

[134] A. Hallak, D. Di-Castro, and S. Mannor, "Model selection in markovian processes," in *International Conference on Machine Learning (ICML)*, 2013.

[135] H. Chen, F. Tang, P. Tino, and X. Yao, "Model-based kernel for efficient time series analysis," in *International Conference on Knowledge Discovery and Data mining (SIGKDD)*, 2013, pp. 392–400.

[136] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *International Conference on Machine Learning (ICML)*, 2001, pp. 282–289.

[137] S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell, "Hidden conditional random fields for gesture recognition," in *Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2006, pp. 1521–1527.

[138] A. Kotsifakos, "Case study: Model-based vs. distance-based search in time series databases," in *Exploratory Data Analysis (EDA) Workshop in SIAM International Conference on Data Mining (SDM)*, 2014.

[139] F. Pachet, G. Ramalho, and J. Carrive, "Representing temporal musical objects and reasoning in the MusES system," *Journal of New Music Research*, vol. 25, no. 3, pp. 252–275, 1996.

[140] P. Kam and A. W. Fu, "Discovering temporal patterns for interval-based events." in *International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, 2000, pp. 317–326.

[141] J. M. Ale and G. H. Rossi, "An approach to discovering temporal association rules," in *Symposium On Applied Computing (SAC)*, 2000, pp. 294–300.

[142] F. Mörchen, "Unsupervised pattern mining from symbolic temporal data," *International Conference on Knowledge Discovery and Data mining (SIGKDD) Exploration Newsletter*, vol. 9, pp. 41–55, June 2007.

[143] F. Mörchen and D. Fradkin, "Robust mining of time intervals with semi-interval partial order patterns," in *SIAM International Conference on Data Mining (SDM)*, 2010, pp. 315–326.

[144] D. Patel, W. Hsu, and M. Lee, "Mining relationships among interval-based events for classification," in *Special Interest Group on Management of Data (SIGMOD)*, 2008, pp. 393–404.

[145] O. Kostakis, P. Papapetrou, and J. Hollmén, "Distance measure for querying arrangements of temporal intervals," in *PErvasive Technologies Related to Assistive Environments (PETRA)*, 2011.

[146] ——, "Artemis: Assessing the similarity of event-interval sequences," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECLM/PKDD)*, 2011, pp. 229–244.

[147] A. Kotsifakos, P. Papapetrou, and V. Athitsos, "IBSM: Interval-based sequence matching," in *SIAM International Conference on Data Mining (SDM)*, 2013, pp. 596–604.

[148] J.-L. Lin, "Mining maximal frequent intervals," in *Symposium On Applied Computing (SAC)*, 2003, pp. 624–629.

[149] R. Villafane, K. A. Hua, D. Tran, and B. Maulik, "Knowledge discovery from series of interval events," *Journal of Intelligent Information Systems (JIIS)*, vol. 15, no. 1, pp. 71–89, 2000.

[150] F. Giannotti, M. Nanni, and D. Pedreschi, "Efficient mining of temporally annotated sequences," in *SIAM Data Mining Conference (SDM)*, vol. 124, 2006, pp. 348–359.

[151] S.-Y. Hwang, C.-P. Wei, and W.-S. Yang, "Discovery of temporal patterns from process instances," *Computers in Industry*, vol. 53, no. 3, pp. 345–364, 2004.

[152] T. Abraham and J. F. Roddick, "Incremental meta-mining from large temporal data sets," in *Advances in Database Technologies*, 1999, pp. 41–54.

[153] X. Chen and I. Petrounias, "Mining temporal features in association rules," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECLM/PKDD)*. Springer-Verlag, 1999, pp. 295–300.

[154] F. Höppner, "Discovery of temporal patterns - learning rules about the qualitative behaviour of time series," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECLM/PKDD)*, 2001, pp. 192–203.

[155] F. Höppner and F. Klawonn, "Finding informative rules in interval sequences," in *International Symposium on Intelligent Data Analysis (IDA)*, 2001, pp. 123–132.

[156] C. Mooney and J. F. Roddick, "Mining relationships between interacting episodes," in *SIAM International Conference on Data Mining (SDM)*, 2004.

[157] S. Laxman, P. Sastry, and K. Unnikrishnan, "Discovering frequent generalized episodes when events persist for different durations," *Transactions on Knowledge and Data Engineering (TKDE)*, vol. 19, no. 9, pp. 1188–1201, 2007.

[158] E. Winarko and J. F. Roddick, "Armada - an algorithm for discovering richer relative temporal association rules from interval-based data," *Data & Knowledge Engineering (DKE)*, vol. 63, no. 1, pp. 76–90, 2007.

[159] P. Papapetrou, G. Benson, and G. Kollios, "Discovering frequent poly-regions in DNA sequences," in *International Conference on Data Mining (ICDM) Workshop on Data Mining in Bioinformatics*, 2006.

[160] S.-Y. Wu and Y.-L. Chen, "Mining nonambiguous temporal patterns for interval-based events," *Transactions on Knowledge and Data Engineering (TKDE)*, vol. 19, no. 6, pp. 742–758, 2007.

[161] Y.-C. Chen, W.-C. Peng, and S.-Y. Le, "CEMiner- an effcient algorithms for mining closed patterns from interval-based data," in *International Conference on Data Mining (ICDM)*, 2011.

[162] F. Mörchen, "Temporal pattern mining in symbolic time point and time interval data," in *International Conference on Knowledge Discovery and Data mining (SIGKDD)*, 2010.

[163] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM (CACM)*, vol. 26, no. 11, pp. 832–843, 1983.

[164] J. Allen and G. Ferguson, "Actions and events in interval temporal logic," *Journal of Logic and Computation (JLP)*, 1994.

[165] D. Patel, W. Hsu, and M. Lee, "Mining relationships among interval-based events for classification," in *Special Interest Group on Management of Data (SIGMOD)*, 2008, pp. 393–404.

BIOGRAPHICAL STATEMENT

Alexios Kotsifakos was born in Athens, Greece, in 1985. He received his B.Sc. degree in the Department of Informatics and Telecommunications (DI&T) from the National and Kapodistrian University of Athens (NKUA), Greece, in 2007, being ranked first (8.74/10) and with honors and awards in all four years of studies. He received his M.Sc. in "Advanced Information Systems" from DI&T in 2009, again being ranked first (9.74/10). From 2008 to 2011 he was a Research and Teaching Assistant at the same Department, and from 2010 to 2011 he was also an Instructor in several Greek Institutes of Professional Training giving lectures on Computer Science courses. In April 2014 he received his Ph.D. degree in Computer Science from the University of Texas at Arlington. His research areas include, but are not limited to, similarity search and efficient indexing techniques for sequence databases applied on clustering and classification tasks, and analysis of sensor data.