SIMILARITY MEASURES AND INDEXING METHODS FOR TIME SERIES AND MULTICLASS RECOGNITION

by

ALEXANDRA STEFAN

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

December 2012

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Gautam Das for his invaluable advice during the course of my doctoral studies. I wish to thank my committee members Dr. Farhad Kamangar, Dr. Heng Huang and Dr. Chengkai Li, 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 the years I spent in school, first in Romania, then at Boston University and finally at UTA. I want to thank the members of the IVC group at Boston University and the VLM Lab at UTA, for their help, support, and camaraderie.

I would like to express my deep gratitude to my family who have encouraged and inspired me and supported my undergraduate and graduate studies. Finally, I would like to thank my husband and children for their patience with me as I was going through graduate school.

November 9, 2012

ABSTRACT

SIMILARITY MEASURES AND INDEXING METHODS FOR TIME SERIES AND MULTICLASS RECOGNITION

Alexandra Stefan, Ph.D.

The University of Texas at Arlington, 2012

Supervising Professor: Gautam Das

This thesis investigates the problem of similarity search in multimedia databases. A key application domain of the proposed work is pattern classification, with emphasis on classification of gestures, handshapes, faces, and time series. A secondary application of the proposed work is efficient similarity search in large biological databases of protein sequences.

More specifically, the 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. On the topic of similarity measures, the thesis contributes a method for measuring similarity in a database of videos from American Sign Language (ASL). This method produces promising results towards enabling useful educational applications for the ASL community. A second contribution of the thesis is on the theoretical problem of how to define a useful metric distance measure for time series data. The thesis proposes a novel metric, called MSM (abbreviation for Move-Split-Merge), which has both attractive theoretical properties and competitive classification accuracy on actual data.

With respect to the problem of improving the efficiency of similarity search, the thesis contributes a novel method for recognition of a large number of classes. While many researchers have worked on the topic of how to train good classifiers for this task, the thesis proposes a new perspective by explicitly addressing efficiency. In particular, the thesis shows that, under some conditions, multiclass recognition becomes theoretically equivalent to similarity search, and in that case we can use off-the-shelf similarity indexing methods to significantly speed up multiclass recognition. The thesis also proposes a dimensionality reduction method specifically designed for speeding up similarity search in large string databases. While dimensionality reduction methods are commonly used in vector spaces, our method allows similar techniques to be used for spaces of strings under the edit distance measure.

Thorough experimental evaluation on a variety of datasets demonstrates state-of-the-art performance for the methods that constitute the contributions of the thesis.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The amount of multimedia data that is digitally stored is constantly increasing. Large databases of such data can potentially store vast amounts of information. However, making such information readily accessible to users can be a challenging task. Users typically need to identify specific pieces of information that are the most relevant for a specific task at hand. The challenge lies in automatically identifying, within a large database, the relatively few pieces of information that the user is looking for. To tackle the challenge we need to design methods that achieve both good accuracy and acceptable efficiency. The goals of accuracy and efficiency are oftentimes at odds with each other; more time-consuming algorithms may improve accuracy, but at the cost of making retrieval time unacceptable to the user.

There is no single answer to the question of how to achieve accuracy and efficiency in searching multimedia databases. The answer depends on many factors, including the specific types of data that we are searching (e.g., video, audio, biological sequences), and the type of information that we use to specify the search (e.g., natural language, keywords, examples of patterns we are looking for). A large body of literature exists that describes various methods for different variations of the problem.

In this thesis we focus on the problem of similarity search, where the user specifies to the system examples, and asks the system to retrieve database items that are the most similar to those examples. A key application of similarity search is pattern classification; in that case, the example provided by the user is a pattern that the user wants the system to classify/recognize. The system performs classification of that example based on the class labels of the most similar database patterns.

Similarity search can also be useful in other settings as well, by simply allowing the user to compare the query example to the most similar database matches, and perhaps to browse through additional annotations that the database may store for those matches, such as, e.g., time and place of origin, or notes that other people made. For example, biologists find it useful, when they want to analyze the properties of a protein, to identify the most similar proteins in a large database. These most similar proteins may contain useful information about how the protein evolved, or what functions it may perform.

Within similarity search, 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. With respect to improving accuracy, the thesis work focuses on time series data, and proposes two specific solutions:

- The first solution, which we first described in [4], addresses the question of how to identify similar matches in a database of videos from American Sign Language (ASL). We will see that addressing this problem opens the way for useful educational applications for the ASL community.
- The second solution, originally published in [5], addresses the theoretical problem of how to define a useful metric distance measure for time series data. We propose a novel metric, called MSM (abbreviation for Move-Split-Merge), which has both attractive theoretical properties and competitive classification accuracy on actual data.

With respect to the problem of improving the efficiency of similarity search, the thesis makes the following two contributions:

- The first contribution, published in [6], is a novel method for recognition of a large number of classes. While many researchers have worked on the topic of how to train good classifiers for this task, our method provides a new perspective by explicitly addressing efficiency. We show that, under some conditions, multiclass recognition becomes theoretically equivalent to similarity search, and in that case we can use off-the-shelf similarity indexing methods to significantly speed up multiclass recognition.
- The second contribution is a dimensionality reduction method specifically designed for speeding up similarity search in large string databases. While dimensionality reduction methods are commonly used in vector spaces, our method shows that similar techniques can be used for spaces of strings under the edit distance measure, despite the fundamental differences in structure between the edit distance and L_p metrics used in vector spaces.

In the next chapters we describe in detail the contributions of the thesis, and the experimental evaluation for demonstrating the usefulness and relevance of these contributions.

CHAPTER 2

A SIMILARITY MEASURE FOR LARGE VOCABULARY SIGN SEARCH

This chapter focuses on a specific application, namely helping users look up the meaning of a sign in American Sign Language (ASL). Looking up the meaning of a sign is not a straightforward task. ASL dictionaries typically allow look-up of ASL signs based on their English translations; that is, these dictionaries are really English to ASL dictionaries, which makes it difficult to look up a sign if the user either does not know the meaning of that sign, or does not know its translation into English. There are ASL dictionaries that allow access to ASL signs based on their articulatory properties, such as handshape [7], but these interfaces either require specification of many articulatory parameters, or else they require the user to scan long lists of signs (for example that share a particular handshape). These lookup methods may fail entirely if the signer is looking for a variant that is different in a small way from the dictionary entry or errs with respect to the specification of the articulatory parameter(s). A system that helps users look up unknown signs would be useful to the millions of users and learners of sign languages around the world (estimated 0.5 to 2 million users in the US [8, 9]). The capability to look up signs would be particularly useful to students of a sign language, as useful as it is for students of a written language (such as English) to be able to look up the meaning of unknown words.

In our approach, having encountered an unknown sign, the user can simply perform the sign in front of a webcam. Then, the system compares the input sign with videos of signs stored in the system database, and presents the most similar signs (and potentially also their English translations) to the user. The user can then view the results and decide which (if any) of those results is correct.

In order to produce a system that works well enough for public use, we have opted for a not fully automatic system, which requires knowing the bounding box of the hands in each frame of both the test and the database videos. For our dataset, we have chosen videos from the public ASLLVD resource [2], where such hand locations are provided for thousands of examples of signs. In our demo system, the user specifies/verifies hand locations, in collaboration with a semi-automated hand detector. Making hand detection more or entirely automatic is a challenging task that we have left for future work.

In our dataset, we have examples from a large vocabulary of 1,113 distinct sign classes. A key constraint is that we only have two training examples for each sign. Given the small number of examples per sign, we use an exemplar-based method, as opposed to a model-based method, such as Hidden Markov Models (HMMs). We start with a baseline similarity measure based on the popular dynamic time warping (DTW) distance [10]. DTW is applied on time series of feature vectors based on hand motion. We improve this baseline similarity measure by incorporating information from hand appearance.

We evaluate our approach in user-independent experiments with a system vocabulary of 1,113 signs. The correct sign was included in the top 10 matches for 78% of the test queries. By considering more signs per query, the user can successfully look up an even larger percentage of query signs. These results are a significant improvement over results previously reported in the literature for comparable vocabulary sizes and under user-independent settings.

2.1 Related Work

Several methods exist for recognizing isolated gestures or signs, as well as continuous signing. The majority of existing methods are model-based, using Hidden Markov Models [11, 12, 13, 14] or alternative approaches such as recursive partition trees [15], boosted volumetric features [16], and hidden conditional random fields [17]. Such methods typically use ten or more training examples per gesture or sign. In contrast, in our setting, we have only two training examples per sign.

Using more examples per sign typically improves accuracy (see, e.g., [18, 19]), but may not be an option, due to lack of data. For example, the Gallaudet dictionary of ASL [20] includes 3,000 signs, and the only public video dataset currently available for a vocabulary of that size is the ASLLVD resource [2], where only two examples are available for most of the signs. Cooper et al. [21] aim at automatically generating large corpora by automatically segmenting signs from close-captioned sign language videos, but the usability of such automatically built corpora as training data was not evaluated. Another promising approach for limited numbers of examples per sign is transfer learning [22], but that approach has only been evaluated in a user-dependent scenario, where the test signs are performed by a user who has also provided data for training.

Exemplar-based approaches offer an alternative when only limited examples per class are available. Motion energy images [23] are a well-known exemplar-based approach, but perform poorly in our experiments. Gorelick et al. [24] represent videos of gestures/actions using 3D shapes extracted by identifying areas of motion in each video frame. However, applying that method to our setting would require accurate silhouette extraction of the hands, which is a challenging task even if the bounding box of the hand is known, especially when hands overlap with each other or with the face, or when the background is cluttered.

Some researchers have reported results on vocabularies of thousands of signs, using input from digital gloves, e.g., [25]. On the other hand, most existing vision-based approaches have been evaluated with vocabularies of some tens of signs, e.g., [11, 15, 13]. Kadir et al. [18] report results on 164 signs, with about 85% accuracy when only two training examples per sign are used, whereas Zieren et al. [19] use a vocabulary of 232 signs, and achieve a remarkable 99.3% accuracy rate. However, in both [18] and [19], a single user signed all the training and test examples. In Zieren et al. [19], when experiments are performed in a user-independent setting, the recognition rate drops from 99.3% to 44%, a drop that highlights the difficulty of user-independent sign recognition.

In earlier work [2, 1] we have reported results on data from the public ASLLVD resource, with vocabulary sizes of 992 and 921 signs respectively, and using methods based on motion energy images in [2] and dynamic time warping (with user-aided hand detection, as in our system) in [1]. In our experiments, the method described here outperforms our earlier approaches [2, 1] by a large margin.

2.2 Application Overview

When a user encounters an unknown sign, the user can perform the sign in front of a webcam, or submit an existing video of that sign. Then, the system asks the user to mark the start and end frames of the actual sign in the video, and to indicate whether the sign is one-handed or two-handed (see Figure 2.1), and which is the dominant hand (if there is an asymmetry in the production of the sign).

At the next step, the system detects bounding boxes of hands on all frames using features based on skin color and motion. The user views the hand detection results, and can correct those results on any frame. As soon as the user makes a correction, the system propagates information from that correction to improve the detection results in the rest of the frames.

After hand detection results have been approved by the user, the system computes the similarity between the query sign and all database signs. The system ranks the 1,113 distinct signs in decreasing order of similarity to the query. There are two examples for each of 1,113 distinct signs in the database, thus producing two similarity scores with respect to the query. For the purposes of ranking the signs, the better of those two scores is kept.

Once the signs have been ranked, the system presents to the user an ordered list of the best matching signs. The user then views the results, starting from the highest-ranked sign, until encountering the video displaying the actual sign of interest. For example, if the correct match was ranked as 5th best by the system, the user needs to view the top five results in order to view the correct match.

When the user identifies the correct database sign, the user can readily view any additional information associated with that sign. Currently, our signs are labeled with very rough English glosses. These do not necessarily provide accurate information about the meaning of the signs, however, since there is no 1-1 relationship between ASL signs and English words. The longer term plan is that this interface may provide access to sophisticated multi-media ASL language resources, which would provide more extensive information about the signs being looked up. For the time being, though, the user may find the very rough translation to be of some utility.

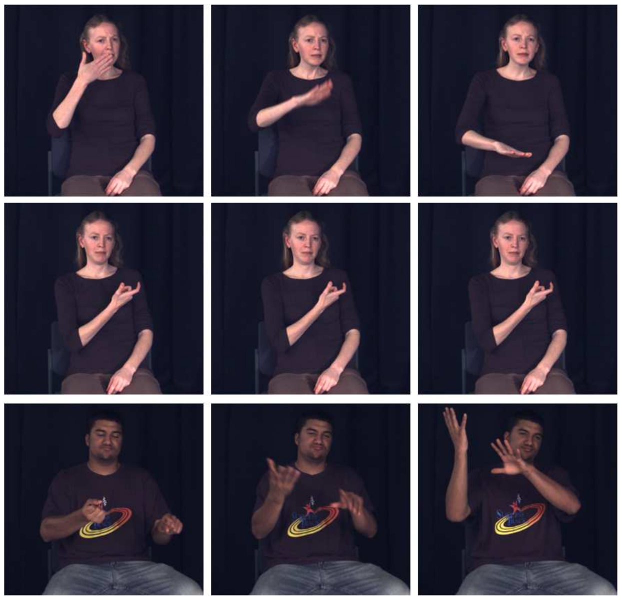


Figure 2.1. Examples of three signs from the dataset that we use. For each sign, we show the first, middle, and last frame. Top row: a one-handed sign meaning “bad”. Middle row: a one-handed sign meaning “badge”, and exhibiting only little motion. Bottom row: a two-handed sign meaning “abandon”..

2.2.1 Measures of Accuracy

As far as the user is concerned, the system has succeeded on a query sign if the user has indeed managed to retrieve the sign that was being sought. One possible type of failure is a situation in which the query sign is not part of the system vocabulary. For the purposes of this paper we ignore this source of failure, as it does not depend on the quality of the underlying technology, but simply on the size of the database corpus.

A second type of failure results when the system ranks the correct match too low. A user would probably not be willing to view more than the top 10 or 20 signs from the results, although this may vary across users. If a user is willing to view at most k results, then the system fails when the correct match is not among those top k results.

Consequently, given a query Q, a key measure of performance is the rank $R(Q)$ that the system assigns to the correct result for Q. Given an integer k, we define a boolean measure of success $S(Q,k)$, that is true iff $R(Q) \leq k$. The success rate $S(k)$ over a test set $\mathbb{Q}$ of queries is simply the average success rate $S(Q,k)$ over $\mathbb{Q}$. For notational convenience we also define $K(s)$ to be (loosely) an inverse function of S: given a desired success rate s, $K(s)$ is the maximum number of results per query that the user must consider to obtain that success rate, so that $K(S(k)) = k$. More formally:

$$\begin{aligned}
R(Q) &= \text{the rank the system assigns to the correct result for } Q \ . && (2.1)\\
S(Q,k) &= \begin{cases} 1 & \text{if } R(Q) \leq k \ . \\ 0 & \text{otherwise} \ . \end{cases} && (2.2)\\
S(k) &= \text{mean}\{S(Q,k) \mid Q \in \mathbb{Q}, \text{where } \mathbb{Q} \text{ is a test set.}\} \ . && (2.3)\\
K(s) &= \text{the } k \text{ such that } S(k) = s \ . && (2.4)
\end{aligned}$$

Even if the user is willing to view $K(s)$ results per query, if the correct match is ranked at $R(Q) < K(s)$, then the user can stop viewing results as soon as the user encounters the correct match. Consequently, another meaningful measure of accuracy is the *average* number of results that the user needs to consider per query until encountering the correct result, for a given success rate s. We define that measure as $A(s)$:

$$A(s) = \text{mean}\{R(Q) \mid (Q \in \mathbb{Q}) \wedge (R(Q) \leq K(s))\} \ . \qquad (2.5)$$

2.3 Features and Normalization

Let X be a video of a sign. We denote by $|X|$ the number of frames in the video, and by $X(t)$ the t-th frame of that video, t ranging from 1 to $|X|$. From sign X we extract the following location, orientation, and hand appearance features:

- $L_d(X,t)$ and $L_{nd}(X,t)$: The (x,y) centroid respectively of the dominant hand and non-dominant hand of the signer at frame t.
- $L_\delta(X,t)$: The relative position of the dominant hand with respect to the non-dominant hand at frame t. $L_\delta(X,t) = L_d(X,t) - L_{nd}(X,t)$.
- $O_d(X,t)$ and $O_{nd}(X,t)$: The unit vectors representing the direction of motion from $L_d(X,t-1)$ to $L_d(X,t+1)$ and from $L_{nd}(X,t-1)$ to $L_{nd}(X,t+1)$.
- $O_\delta(X,t)$: The unit vector representing the direction of motion from $L_\delta(X,t-1)$ to $L_\delta(X,t+1)$.
- $H_{d,s}(X), H_{d,e}(X), H_{nd,s}(X)$, and $H_{nd,e}(X)$: images of the dominant and the non-dominant hand at the start and end frame of the video.

Each hand appearance image H is preprocessed using the following steps:

1. We start by simply cropping the subwindow H_1 corresponding to the bounding box of the hand.
2. We detect skin in that window, using the method of Jones et al. [26].
3. We set all non-skin pixels in H_1 to 0.
4. We create H_2 to be the grayscale version of H_1.
5. We normalize H_2 to have a mean of zero and a standard deviation of 1.
6. We create H_3 as a scaled version of H_2, so that the longest side of H has length 50.
7. The final image H is a padded version of H_3, to make sure that H has an equal number of rows and columns. Additional rows or columns are added as needed,

with values of zero. The padding is applied symmetrically, so that the centroid of the hand corresponds with the center of the final image H.

For notational convenience, all features referring to the non-dominant hand (i.e., $L_{nd}, L_{\delta}, O_{nd}, O_{\delta}, H_{nd,s}, H_{nd,e}$) are set to zero vectors for one-handed signs.

2.3.1 Coordinate System

In defining location features, the choice of coordinate system is important. To account for differences in translation and spatial scale between the query video and the matching training videos, we use a face-centric coordinate system. We use the face detector of Rowley et al. [27] to detect the face of each signer at the first frame of the sign. The coordinate system is defined so that the center of the face is at the origin, and the diagonal of the face bounding box has length 1. The same scaling factor is applied to both the x and the y direction. Features L_d, L_{nd}, L_{δ} are all defined in this normalized coordinate system.

2.3.2 Time Series Length Normalization

Different signers may sign at different speeds. Dynamic Time Warping (DTW), which we describe in Section 2.4, is a similarity measure that is biased against longer database matches, and this bias is more noticeable for short queries. To account for that, we normalize each sequence, so that the length of all sequences is the same (20 in our experiments). In particular, we resample the sequences of L_d, L_{nd}, and L_{δ} features extracted from each sign, so that each sequence has length 20. Resampling is done using linear interpolation. As shown in our experiments, this normalization significantly improves accuracy.

2.4 Comparing Trajectories via Dynamic Time Warping

Let X be a video of a sign. We can represent X as a time series $(X_1, \ldots, X_{|X|})$, where each X_t is simply a concatenation of the features extracted at frame t:

$$X_t = (L_d(X,t), L_{nd}(X,t), L_\delta(X,t), O_d(X,t), O_{nd}(X,t), O_\delta(X,t)) \; . \tag{2.6}$$

As a reminder, features $L_{nd}, L_\delta, O_{nd}, O_\delta$ are set to 0 for one-handed signs.

Dynamic Time Warping (DTW) [10] is a commonly used distance measure for time series. Given two sign videos Q and X, DTW computes a warping path W establishing correspondences between frames of Q and frames of X:

$$W = ((q_1, x_1), \ldots, (q_{|W|}, x_{|W|})) \; , \tag{2.7}$$

where $|W|$ is the length of the warping path, and pair (q_i, x_i) means that frame q_i of Q corresponds to frame x_i of X. A warping path must follow two constraints:

- boundary constraints: $q_1 = 1,\ x_1 = 1,\ q_{|W|} = |Q|,\ x_{|W|} = |X|$.
- monotonicity and continuity: $0 \leq q_{i+1} - q_i \leq 1, \quad 0 \leq x_{i+1} - x_i \leq 1$.

The cost $C(W, Q, X)$ of a warping path W is the sum of individual local costs $c(Q_{q_i}, X_{x_i})$, corresponding to matching each Q_{q_i} with the corresponding X_{x_i}:

$$C(W, Q, X) = \sum_{i=1}^{|W|} c(Q_{q_i}, X_{x_i}) \; . \tag{2.8}$$

As local cost c, we use a weighted linear combination of the individual Euclidean distances between the six features extracted from the two frames:

$$\begin{aligned} c(Q_{q_i}, X_{x_i}) \;=\; & f_1\|L_d(Q,q_i) - L_d(X,x_i)\| + f_2\|L_{nd}(Q,q_i) - L_{nd}(X,x_i)\| + \\ & f_3\|L_\delta(Q,q_i) - L_\delta(X,x_i)\| + f_4\|O_d(Q,q_i) - O_d(X,x_i)\| + \\ & f_5\|O_{nd}(Q,q_i) - O_{nd}(X,x_i)\| + f_6\|O_\delta(Q,q_i) - O_\delta(X,x_i)\| \end{aligned} \tag{2.9}$$

In the above equation, $\|\cdot\|$ stands for the Euclidean norm. In our experiments, weights f_j are optimized using cross-validation on the training set.

The DTW distance $D_{\text{DTW}}(Q, X)$ between sign videos Q and X is defined as the cost of the lowest-cost warping path between Q and X:

$$D_{\text{DTW}}(Q, X) = \min_{W} C(W, Q, X) \tag{2.10}$$

The optimal warping path and the distance $D_{\text{DTW}}(Q, X)$ can be computed using dynamic programming, with a time complexity of $O(|Q||X|)$ [10].

2.5 Incorporating Hand Appearance

The D_{DTW} distance measure defined above depends only on the trajectories of the two hands. At the same time, the appearance of the hand is an important additional source of information about the identity of a sign. Recognizing handshape is a challenging task, especially when a hand appears in front of another skin-colored object such as the other hand or the face. Given the difficulty of this topic, we have postponed the task of implementing a sophisticated similarity measure for hand appearance for future work. Instead, in this paper we have opted for the simplest possible option, which is the Euclidean distance between hand appearance images. Despite its simplicity, this approach has led to significant improvements in accuracy, as shown in the experiments.

In particular, we define a distance $D_{\text{hand}}(Q, X)$ between two sign videos Q and X as follows:

$$\begin{aligned} D_{\text{hand}}(Q, X) = {} & \|H_{d,s}(Q) - H_{d,s}(X)\| + \|H_{d,e}(Q) - H_{d,e}(X)\| + \\ & \|H_{nd,s}(Q) - H_{nd,s}(X)\| + \|H_{nd,e}(Q) - H_{nd,e}(X)\| \, . \end{aligned} \tag{2.11}$$

As a reminder (see Section 2.3), each hand image has been preprocessed, by scaling/padding to a canonical size, and removing non-skin pixels.

Combining D_{DTW} and D_{hand} can be done by simply taking a weighted sum of the two distances:

$$D(Q, X) = D_{\text{DTW}}(Q, X) + f_{\text{hand}} D_{\text{hand}}(Q, X) \ . \tag{2.12}$$

The weight f_{hand} is chosen by searching over many possible values, so as to optimize performance on the training data.

2.6 Experiments

Our dataset includes 1,113 distinct sign classes. For each sign class there are three examples, each from a different user. All sign videos and annotations have been downloaded from the ASLLVD website [2]. Although the ASLLVD website includes four synchronized camera views for each sign, only a single frontal view is used for each sign in our experiments.

The dataset was divided into three groups, each group containing a single example from each of the 1,113 classes. Experiments were performed in a user-independent manner, by ensuring that each signer appeared in only a single group out of those three groups. Each group was in turn used as the test set, with the other two groups used as training. All experimental measurements are averaged over the three test groups. All weights involved in defining the overall distance measure were computed exclusively from the training data, and thus a different combination of weights was applied for each test group.

We use the measures of accuracy defined in Section 2.2.1, and in particular $K(s)$ and $A(s)$, which are respectively the maximum and average number of results that a user must consider for a query in order to encounter the correct result for a fraction s of all queries.

Since the user indicates at query time whether a sign is one-handed or two-handed, signs using a different number of hands than the query are not considered for that query. (For real use cases in the longer term, we will have to allow a small probability for a canonically 2-handed sign being produced with just 1 hand, and for a 1-handed sign to be produced with 2 hands.) Signs performed with the left hand as the dominant hand are replaced by mirrored versions, so that we can treat all database and query signs as right-handed. These rules have been applied in all experiments for all methods.

2.6.1 Results

We compare our method to previous methods applied to data from the ASLLVD dataset, namely to the approach of using motion energy images (MEI) described in [2], and the DTW-based approach reported in [1]. With respect to the DTW method of [1], we should note that it corresponds to a stripped-down version of our method, which only uses the L_d and L_{nd} features, does not normalize the length of all time series to a fixed constant, and does not use hand appearance.

Figure 2.2 shows comparative results for the three methods. As measures of accuracy, we use functions $K(s)$ and $A(s)$ (defined in Section 2.2.1), which are, respectively, the maximum and average number of results per query that a user must consider in order to encounter the correct result for a fraction s of all queries. Our method clearly outperforms the two other methods. As an example, the percentage of queries for which the the correct sign is not included in the top 25 results is, respectively, 66% for MEI, 38.6% for Stefan et al. [1], and 11.9% for our method. Similarly, the correct sign is successfully included in the top 10 results for only 20.4% of the queries using MEI, for 47.8% of the queries using the method of Stefan et al. [1], and for 78.4% of queries for our method. For our method, for a success rate of 78.4% the user needs to consider at most 10, and on average 2.36 results per query, until encountering the correct result.

In Figure 2.3 we evaluate three different variations of our method: the first variation, denoted as “DTW without length normalization”, does not use hand appearance and also does not use the resampling step described in Section 2.3.2, which normalizes all time series to length 20. The second variation, denoted as DTW, does not use hand appearance. The third variation, denoted as “DTW + hands”, is the full method described in this paper, that incorporates information from both DTW and hand appearance. As see from the results, normalizing all time series to the same length significantly improves accuracy, and incorporating hand appearance leads to a noticeable additional improvement.

In terms of running time, the system takes on average about one second to compare a query video to the 2,226 database videos. Running time was measured on a PC with an Intel quad-core CPU, running at 2.4GHz, and with 3GB of memory. Our method has been implemented as a single-threaded application.

2.7 Discussion and Future Work

This chapter has presented a method for helping users look up unknown signs, using similarity-based retrieval in a database containing examples of signs from a large vocabulary. In our method, feature vectors are defined based on hand motion and hand appearance. Similarity between signs is measured by combining dynamic time warping scores, which are based on hand motion, with Euclidean distances between hand appearances.

There are several research topics that will be interesting to pursue as future work, with the goal of further improving system performance and the overall user experience. While in the current system hand detection is only semi-automatic, a more (or entirely) automated hand detector will significantly enhance the user experience. Also, while our simple way of using hand appearance led to good results, there is clearly room for improvement in how we use hand appearance, and that is another topic that will be interesting to explore. Our current approach of not allowing one-handed signs to be matched with two-handed signs, and of requiring the user to specify the dominant hand for the query sign, has limitations that need to be addressed. Finally, although the proposed approach works reasonably well in our experiments, we believe that more work is needed in order to satisfactorily address the question of how to learn a good similarity measure for a large vocabulary of signs, given only one or two training examples per sign.

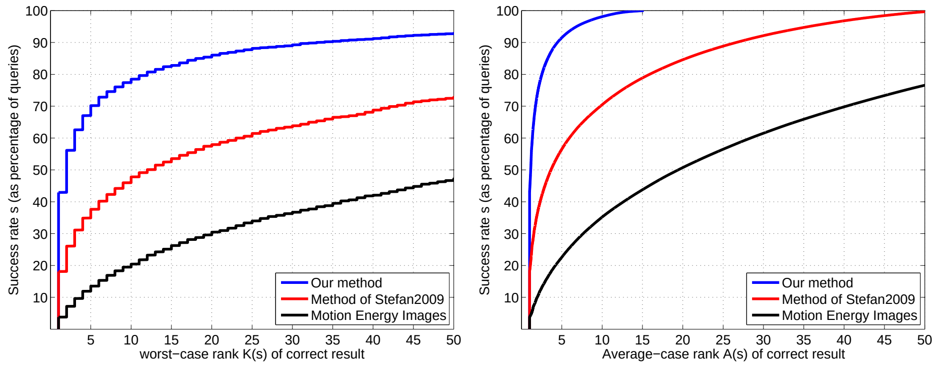


Figure 2.2. Comparison of our method, the method described by Stefan et al. in [1], and the MEI-based method used in [2]. The y-axis corresponds to success rates s. The x-axis corresponds to values of $K(s)$ on the left, and to values of $A(s)$ on the right..

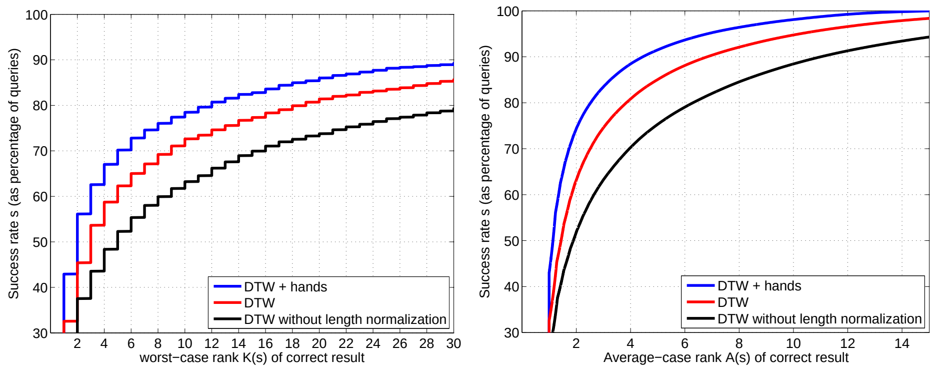


Figure 2.3. Comparison of DTW without length normalization, DTW, and our full method that combines DTW scores with hand appearance similarity scores. The y-axis corresponds to success rates s. The x-axis corresponds to values of $K(s)$ on the left, and to values of $A(s)$ on the right..

CHAPTER 3

THE MOVE-SPLIT-MERGE METRIC FOR TIME SERIES

Time series data naturally appear in a wide variety of domains. Chapter 2 has described in detail one such domain, namely recognition of signs in sign language videos. Other domains include financial data (e.g. stock values), scientific measurements (e.g. temperature, humidity, earthquakes), medical data (e.g. electrocardiograms), audio, video, and human activity representations. Large time series databases can serve as repositories of knowledge in such domains, especially when the time series stored in the database are annotated with additional information such as class labels, place and time of occurrence, causes and consequences, etc.

A key design issue in searching time series databases is the choice of a similarity/distance measure for comparing time series. In this chapter, we describe a novel metric for time series, called MSM (move-split-merge). The key idea behind the MSM metric is to define a set of operations that can be used to transform any time series into any other time series. Each operation incurs a cost, and the distance between two time series X and Y is the cost of the cheapest sequence of operations that transforms X into Y.

The MSM metric uses as building blocks three fundamental operations: Move, Split, and Merge. A Move operation changes the value of a single point of the time series. A Split operation splits a single point of the time series into two consecutive points that have the same value as the original point. A Merge operation merges two consecutive points that have the same value into a single point that has that value. Each operation has an associated cost. The cost of the Move operation is the absolute difference between the old value and the new value. The cost of each Split and Merge operation is equal and set to a constant.

Our main motivation in formulating MSM has been to satisfy, with a single distance measure, a set of certain desirable properties that no existing method satisfies fully. One such property is robustness to temporal misalignments. Such robustness is entirely lacking in methods where time series similarity is measured using the Euclidean distance [28, 29, 30, 31] or variants [32, 33, 34]. Such methods cannot handle even the smallest misalignment caused by time warps, insertions, or deletions.

Another desired property is metricity. As detailed in Section 3.2.1.1, metricity allows the use of an extensive arsenal of generic methods for indexing, clustering, and visualization, that have been designed to work in any metric space.

Several distance measures based on dynamic programming (DP), while robust to temporal misalignments, are not metric. Such methods include dynamic time warping (DTW) [10], constrained dynamic time warping (cDTW) [35], Longest Common Subsequence (LCSS) [36], Minimal Variance Matching (MVM) [37], and Edit Distance on Real Sequence (EDR) [38]. All those measures are non-metric, and in particular do not satisfy the triangle inequality.

Edit Distance with Real Penalty (ERP) [39] is a distance measure for time series that is actually a metric. Inspired by the edit distance [40], ERP uses a sequence of "edit" operations, namely insertions, deletions, and substitutions, to match two time series to each other.

However, ERP has some behaviors that, in our opinion, are counterintuitive. First, ERP is not translation-invariant: changing the origin of the coordinate system changes the distances between time series, and can radically alter similarity rankings. Second, the cost of inserting or deleting a value depends exclusively on the absolute magnitude of that value. Thus, ERP does not treat all values equally; it explicitly prefers inserting and deleting values close to zero compared to other values. In our formulation, we aimed to ensure both translation invariance and equal treatment of all values.

A desired property of any similarity measure is computational efficiency. Measuring the DTW or ERP distance between two time series takes time quadratic to the sum of lengths of the two time series, whereas linear complexity is achieved by the Euclidean distance and, arguably, cDTW (if we treat the warping window width, a free parameter of cDTW, as a constant).

One of our goals in designing a new distance measure was to not significantly exceed the running time of DTW and ERP, and to stay within quadratic complexity.

The proposed MSM metric is our solution to the problem of satisfying, with a single measure, the desired properties listed above: robustness to misalignments, metricity, translation invariance, treating all values equally, and quadratic time complexity. The MSM formulation deviates significantly from existing approaches, such as ERP and DTW, and has proven quite challenging to analyze. While the proposed algorithm is easy to implement in a few lines of code (see Figure 3.10), proving that these few lines of code indeed compute the correct thing turned out to be a non-trivial task, as shown in Sections 3.3 and 3.4. We consider the novelty of the formulation and the associated theoretical analysis to be one of the main contributions of this paper.

For real-world applications, satisfying all the above-mentioned properties would be of little value, unless the distance measure actually provides meaningful results in practice. Different notions of what is meaningful may be appropriate for different domains. At the same time, a commonly used measure of meaningfulness is the nearest neighbor classification error rate attained in a variety of time series datasets. We have conducted such experiments using the UCR repository of time series datasets

[3]. The results that we have obtained illustrate that MSM performs quite well compared to existing competitors, such as DTW and ERP, yielding the lowest error rate in several UCR datasets.

3.1 Defining the MSM Distance

Similar to the edit distance and ERP, MSM uses a set of operations that can transform any time series to any other time series. The basic operations in the edit distance and ERP are Insert, Delete, Substitute. MSM also uses the Substitute operation, we just have renamed it and call it the Move operation. This operation is used to change one value into another.

Our point of departure from the edit distance and ERP is in handling insertions and deletions. In the edit distance, all insertions and deletions cost the same. In ERP, insertions and deletions cost the absolute magnitude of the value that was inserted or deleted. Instead, we aimed for a cost model where inserting or deleting a value depends on both that value and the adjacent values. For example, inserting a 10 between two 10s should cost the same as inserting a 0 between two 0s, and should cost less than inserting a 10 between two 0s.

Our solution is to not use standalone Insert and Delete operations, and instead to use Split and Merge operations. A Split repeats a value twice, and a Merge merges two successive equal values into one. In MSM, an Insert is decomposed to a Split (to create a new element) followed by a Move (to set the value of the new element). Similarly, a delete is decomposed to a Move (to make an element equal in value to either the preceding or the following element) followed by a Merge (to delete the element we just moved). This way, the cost of insertions and deletions depends on the similarity between the inserted or deleted value and its neighbors.

We now proceed to formally define the three basic operations and the MSM distance. Let time series $X = (x_1, \ldots, x_m)$ be a finite sequence of real numbers x_i. The Move operation, and its cost, are defined as follows:

$$\text{Move}_{i,v}(X) = (x_1, \ldots, x_{i-1}, x_i + v, x_{i+1}, \ldots, x_m) \ . \tag{3.1}$$

$$\text{Cost}(\text{Move}_{i,v}) = |v|. \tag{3.2}$$

In words, operation $\text{Move}_{i,v}(X)$ creates a new time series X', that is identical to X, except that the i-th element is *moved* from value x_i to value $x_i + v$. The cost of this move is the absolute value of v.

The Split operation, and its cost, are defined as:

$$\text{Split}_i(X) = (x_1, \ldots, x_{i-1}, x_i, x_i, x_{i+1}, \ldots, x_m) \ . \tag{3.3}$$

$$\text{Cost}(\text{Split}_i) = c. \tag{3.4}$$

Operation $\text{Split}_i(X)$ creates a new time series X', obtained by taking X and splitting the i-th element of X into two consecutive elements. The cost of this split is a nonnegative constant c, which is a system parameter.

The Merge operation acts as the inverse of the Split operation. The Merge operation is invoked as $\text{Merge}_i(X)$, and is only applicable if $x_i = x_{i+1}$. Given a time series $X = (x_1, \ldots, x_m)$, and assuming $x_i = x_{i+1}$:

$$\text{Merge}_i(X) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) \ . \tag{3.5}$$

$$\text{Cost}(\text{Merge}_i) = c \ . \tag{3.6}$$

Operation $\text{Merge}_i(X)$ creates a new time series X', that is identical to X, except that elements x_i and x_{i+1} (which are equal in value) are *merged* into a single element. The cost of a Merge operation is equal to the cost of a Split operation.

This is necessary, as we explain in Section 3.1, in order for MSM to be metric (otherwise, symmetry would be violated).

Figures 3.1 and 3.4 show example applications of Move, Split, and Merge operations.

We define a transformation $\mathbb{S} = (S_1, \ldots, S_{|\mathbb{S}|})$ to be a sequence of operations, where $|\mathbb{S}|$ indicates the number of elements of $\mathbb{S}$. Each S_k in the transformation $\mathbb{S}$ is some Move_{i_k,v_k}, Split_{i_k}, or Merge_{i_k} operation, for some appropriate values for i_k and v_k. The result of applying transformation $\mathbb{S}$ to time series X is the result of consecutively applying operations $S_1, \ldots, S_{|\mathbb{S}|}$ to X:

$$\text{Transform}(X, \mathbb{S}) = \text{Transform}(S_1(X), (S_2, ..., S_{\|\mathbb{S}\|})) \ . \tag{3.7}$$

In the trivial case where $\mathbb{S}$ is the empty sequence (), we can define $\text{Transform}(X, ()) = X$.

The cost of a sequence of operations $\mathbb{S}$ on X is simply the sum of costs of the individual operations:

$$\text{Cost}(\mathbb{S}) = \sum_{k=1}^{|\mathbb{S}|} \text{Cost}(S_k) \ . \tag{3.8}$$

Given two time series X and Y, there are infinite transformations $\mathbb{S}$ that transform X into Y. An example of such a transformation is illustrated in Figure 3.4.

Using the above terminology, we are now ready to formally define the MSM distance. The MSM distance $D(X, Y)$ between two time series X and Y is defined to be the cost of the lowest-cost transformation $\mathbb{S}$ such that $\text{Transform}(X, \mathbb{S}) = Y$. We note that this definition does not provide a direct algorithm for computing $D(X, Y)$. Section 3.4 provides an algorithm for computing the MSM distance between two time series.

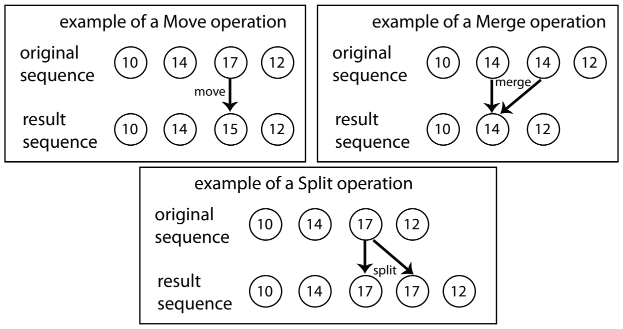


Figure 3.1. Examples of the Move, Split, Merge operations. .

3.2 Motivation for MSM: Metricity and Invariance to the Choice of Origin

In this section we show that the MSM distance satisfies two properties: metricity and invariance to the choice of origin. Satisfying those two properties was a key motivation for our formulation. We also discuss simple examples highlighting how MSM differs from DTW and ERP with respect to these properties.

3.2.1 Metricity

The MSM distance satisfies reflexivity, symmetry, and the triangle inequality, and thus MSM satisfies the criteria for a metric distance. In more detail:

Reflexivity: Clearly, $D(X, X) = 0$, as an empty sequence of operations, incurring zero cost, converts X into itself. If $c > 0$, then any transformation $\mathbb{S}$ that converts X into Y must incur some non-zero cost. If, for some domain-specific reason, it is desirable to set c to 0, an infinitesimal value of c can be used instead, to guarantee reflexivity, while producing results that are practically identical to the $c = 0$ setting.

Symmetry: Let S be a Move, Split, or Merge operation. For any such S there exists an operation S^{-1} such that, for any time series X, $S^{-1}(S(X)) =$ X. In particular:

- The inverse of $\text{Move}_{i,v}$ is $\text{Move}_{i,-v}$.
- Split_i and Merge_i are inverses of each other.

Any sequence of operations $\mathbb{S}$ is also reversible: if $\mathbb{S} = (S_1, \ldots, S_{|\mathbb{S}|})$, then the inverse of $\mathbb{S}$ is $\mathbb{S}^{-1} = (S^{-1}_{|\mathbb{S}|}, \ldots, S^{-1}_1)$. Transform$(X, \mathbb{S}) = Y$ if and only if Transform$(Y, \mathbb{S}^{-1}) = X$.

It is easy to see that, for any operation S, Cost$(S) =$ Cost(S^{-1}). Consequently, if $\mathbb{S}$ is the cheapest (or a tie for the cheapest) transformation that converts X into Y, then $\mathbb{S}^{-1}$ is the cheapest (or a tie for the cheapest) transformation that converts Y into X. It readily follows that $D(X, Y) = D(Y, X)$.

Triangle inequality: Let X, Y, and Z be three time series. We need to show that $D(X, Z) \leq D(X, Y) + D(Y, Z)$. Let $\mathbb{S}_1$ be an optimal (i.e., lowest-cost) transformation of X into Y, so that Cost$(\mathbb{S}_1) = D(X, Y)$. Similarly, let $\mathbb{S}_2$ be an optimal transformation of Y into Z, so that Cost$(\mathbb{S}_2) = D(Y, Z)$. Let's define $\mathbb{S}_3$ to be the concatenation of $\mathbb{S}_1$ and $\mathbb{S}_2$, that first applies the sequence of operations in $\mathbb{S}_1$, and then applies the sequence of operations in $\mathbb{S}_2$. Then, Transform$(X, \mathbb{S}_3) = Z$ and Cost$(\mathbb{S}_3) = D(X, Y) + D(Y, Z)$.

If $\mathbb{S}_3$ is the cheapest (or a tie for the cheapest) transformation converting X into Z, then, $D(X, Z) = D(X, Y) + D(Y, Z)$, and the triangle inequality holds. If $\mathbb{S}_3$ is *not* the cheapest (or a tie for the cheapest) transformation converting X into Z, then $D(X, Z) < D(X, Y) + D(Y, Z)$, and the triangle inequality still holds.

3.2.1.1 Advantages of Metricity

Metricity distinguishes MSM from several alternatives, such as DTW [10], LCSS [36], MVM [37], and EDR [38]. Metricity allows MSM to be combined with an extensive arsenal of off-the-shelf, generic methods for indexing, clustering, and visualization, that have been designed to work in any metric space.

With respect to indexing, metricity allows the use of generic indexing methods designed for arbitrary metrics (see [41] for a review). Examples of such methods

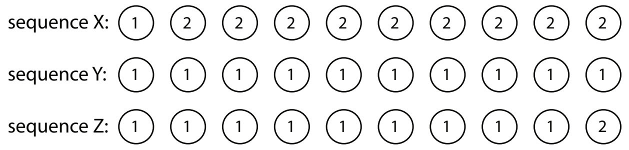


Figure 3.2. An example where DTW violates the triangle inequality: $\mathrm{DTW}(X, Y) = 9$, $\mathrm{DTW}(X, Z) = 0$, $\mathrm{DTW}(Z, Y) = 1$. Thus, $\mathrm{DTW}(X, Z) + \mathrm{DTW}(Z, Y) < \mathrm{DTW}(X, Y)$..

include VP-trees [42] and Lipschitz embeddings [43]. In fairness to competing non-metric alternatives, we should mention that several custom-made indexing methods have been demonstrated to lead to efficient retrieval using non-metric time series distance measures [35, 44, 45].

Another common operation in data mining systems is clustering. Metricity allows the use of clustering methods that have been designed for general metric spaces. Examples of such methods include [46, 47, 48].

Metricity also allows for better data visualization in time series datasets. Visualization typically involves an approximate projection of the data to two or three Euclidean dimensions, using projection methods such as, e.g., MDS [49], GTM [50], or FastMap [51]. In general, projections of non-Euclidean spaces to a Euclidean space, and especially to a low-dimensional Euclidean space, can introduce significant distortion [43]. However, non-metricity of the original space introduces an additional source of approximation error, which is not present if the original space is metric.

As an example, suppose that we want to project to 2D the three time series shown in Figure 3.2, so as to visualize the DTW distances among those three series. Any projection to a Euclidean space (which is metric) will significantly distort the non-metric relationship of those three time series. On the other hand, since MSM is metric, the three MSM distances between the three time series of Figure 3.2 can be captured exactly in a 2D projection.

3.2.1.2 An Example of Non-Metricity in DTW

To highlight the difference between MSM and DTW, Figure 3.2 illustrates an example case where DTW violates the triangle inequality. In that example, the only difference between Y and Z is in the last value, as $y_{10} = 1$ and $z_{10} = 2$. However, this small change causes the DTW distance from X to drop dramatically, from 9 to 0: $DTW(X, Y) = 9$, and $DTW(X, Z) = 0$.

In contrast, in MSM, to transform X into Y, we perform 8 Merge operations, to collapse the last 9 elements of X into a single value of 2, then a single Move operation that changes the 2 into a 1, and 8 Split operations to create 8 new values of 1. The cost of those operations is $16c+1$. To transform X into Z, the only difference is that x_{10} does not need to change, and thus we only need 7 Merge operations, one Move operation, and 7 Split operations. The cost of those operations is $14c + 1$. Thus, the small difference between Y and Z causes a small difference in the MSM distance values: MSM$(Y, Z) = 1$, MSM$(X, Y) = 16c + 1$, MSM$(X, Z) = 14c + 1$.

We should note that, in the above example, constrained DTW (cDTW) would not exhibit the extreme behavior of DTW. However, cDTW is also non-metric, and the more we allow the warping path to deviate from the diagonal, the more cDTW deviates from metric behavior. DTW itself is a special case of cDTW, where the diagonality constraint has been maximally relaxed.

3.2.2 Invariance to the Choice of Origin

Let $X = (x_1, \ldots, x_m)$ be a time series where each x_i is a real number. A translation of X by t, where t is also a real number, is a transformation that adds t to each element of the time series, to produce $X + t = (x_1 + t, \ldots, x_m + t)$. If distance measure D is invariant to the choice of origin, then for any time series X, Y, and any translation t, $D(X,Y) = D(X+t, Y+t)$. The MSM distance is invariant to the choice of origin, because any transformation S that converts X to Y also converts $X+t$ to $Y+t$.

3.2.2.1 Contrast to ERP: Translation Invariance and Equal Treatment of All Values

Invariance to the choice of origin is oftentimes a desirable property, as in many domains the origin of the coordinate system is an arbitrary point, and we do not want this choice to impact distances and similarity rankings. In contrast, the ERP metric [39] is not invariant to the choice of origin. For the full definition of ERP we refer readers to [39].

To contrast MSM with ERP, consider a time series X consisting of 1000 consecutive values of v, for some real number v, and let Y be a time series of length 1, whose only value is a v as well. In ERP, to transform X into Y, we need to delete v 999 times. However, the cost of these deletions depends on the value of v: the cost is 0 if $v = 0$, and is $999v$ otherwise. In contrast, in MSM, the cost of deleting v 999 times (by applying 999 Merge operations) is independent of v. Thus, the MSM distance is translation-invariant (does not change if we add the same constant to both time series), whereas ERP is not.

A simple remedy for making ERP translation-invariant is to normalize each time series so that it has a mean value of 0. However, even in that case, the special treatment of the origin by ERP leads to insertion and deletion costs that are, in our opinion, counterintuitive in some cases. Such a case is illustrated in Figure 3.3. In that example, we define sequence $A = (-1, 0, 1, 0, -1)$. Then, we define sequences B and C, by copying A and inserting respectively a value of -1 and a value of 0 at the

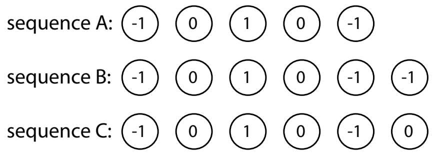


Figure 3.3. An example illustrating the different behavior of MSM and ERP. Both sequences B and C are obtained by inserting one value at the end of A. According to ERP, A is closer to C than to B: $\text{ERP}(A, B) = 1$, $\text{ERP}(A, C) = 0$. According to MSM, A is closer to B than to C: $\text{MSM}(A, B) = c$, $\text{MSM}(A, C) = 1 + c$. .

end. According to ERP, A is closer to C than to B, and actually $ERP(A, C) = 0$, because ERP treats 0 (the origin) as a special value that can be inserted anywhere with no cost. In contrast, according to MSM, A is closer to B, as a single Split operation of cost c transforms A to B. Transforming A to C requires a Split and a Move, and costs $c + 1$.

This difference between MSM and ERP stems from the fact that, in MSM, the cost of inserting or deleting a value v only depends on the difference between v and its adjacent values in the time series. Thus, in MSM, inserting a 10 between two 10's is cheaper (cost $= c$) than inserting a 10 between two zeros (cost $= 10 + c$), and inserting a 10 between two zeros is as expensive (cost $= 10 + c$) as inserting a 0 between two 10s. On the other hand, ERP treats values differently depending on how close they are to the origin: inserting a 10 between two 10's costs the same (cost $= 10$) as inserting a 10 between two zeros, and inserting a 10 between two zeros (cost $= 10$) is more expensive than inserting a 0 between two 10's (cost $= 0$).

3.3 Transformation Graphs and the Monotonicity Lemma

In Section 3.4 we describe an algorithm that computes the MSM distance between two time series. However, the correctness of that algorithm derives from cer-

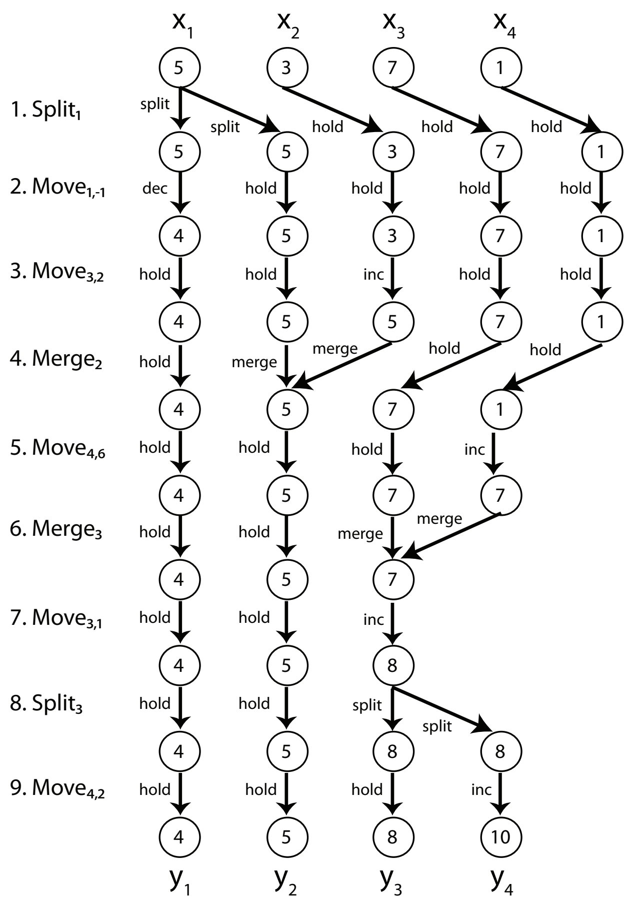


Figure 3.4. An example of a (non-optimal, but easy-to-visualize) transformation that converts input time series $(5, 3, 7, 1)$ into output time series $(4, 5, 8, 10)$. We see the effects of each individual operation in the transformation, and we also see the step-by-step graph defined by applying this transformation to the input time series..

tain theoretical observations. In this section we lay the theoretical groundwork for explaining the algorithm of Section 3.4.

3.3.1 Step-By-Step Graphs and Transformation Graphs

For any time series X and any transformation $\mathbb{S}$ we can draw what we call a step-by-step graph, that illustrates the intermediate results and the final result that we obtain, starting with X, and applying in sequence the operations of transformation $\mathbb{S}$. An example of such a graph is shown in Figure 3.4. In that figure, $X = (5, 3, 7, 1)$,

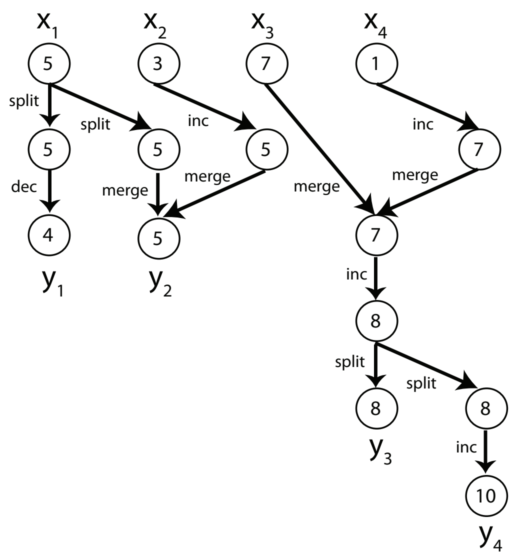


Figure 3.5. The transformation graph corresponding to the step-by-step graph of Figure 3.4..

and transformation $\mathbb{S}$ consists of nine operations, which are shown in detail. The final result of Transform$(X, \mathbb{S})$ is time series $Y = (4, 5, 8, 10)$.

The step-by-step graph is a directed graph, that is divided into layers. The first layer corresponds to the input sequence X. Layer $k + 1$, for $k > 0$ corresponds to the result of applying the first k operations of $\mathbb{S}$ on X. In intermediate layers, every node is connected to one or two parent nodes, and one or two children nodes. Every directed edge has a label that shows how the child node was obtained from the parent node. There are five types of edge labels:

- HOLD: A HOLD edge indicates that no Move, Split, or Merge operation was applied to the parent node.
- INC: An INC edge indicates that a Move operation was applied to the parent node, and that a positive value was added as a result of the move.
- DEC: A DEC edge indicates that a Move operation was applied to the parent node, and that a a negative value was added as a result of the move.

- SPLIT: A Split operation generates two SPLIT edges, going from a parent node to two children nodes.
- MERGE: A Merge operation generates two MERGE edges, going from two parents to a common child.

In a step-by-step graph, most edges are typically HOLD edges. Given a step-by-step graph, we can obtain a significantly more concise graph, called a transformation graph, by applying the following process:

- Copy the original step-by-step graph.
- Delete all HOLD edges.
- Collapse into a single node any set of nodes that, in the original step-by-step graph, were connected by a path consisting exclusively of HOLD edges.

Figure 3.5 shows the transformation graph obtained from the step-by-step graph of Figure 3.4.

The cost of a transformation graph is defined to be the sum of the costs of the operations appearing in that graph. If a transformation $\mathbb{S}$ has G as its transformation graph, then $\mathbb{S}$ and G have the same cost. Similarly, the cost of a path in a transformation graph is defined to be the sum of the costs of the operations associated with the edges of the path.

Any step-by-step graph corresponds to one and only one sequence of operations, because the step-by-step graph imposes a full order on the set of operations appearing in that graph. On the other hand, a transformation graph imposes only a partial order on the set of operations appearing in that graph. Given a transformation graph G, a sequence of operations $\mathbb{S}$ has G as its transformation graph if:

- $\mathbb{S}$ contains all the operations appearing in G.
- $\mathbb{S}$ contains no operation that does not appear in G.
- The order of operations in $\mathbb{S}$ respects the partial order defined by G.

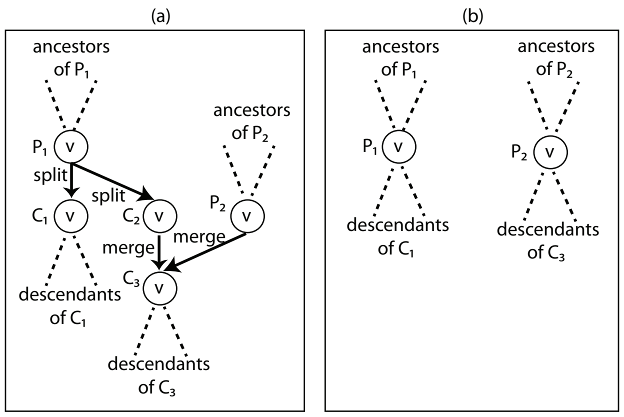


Figure 3.6. Editing a transformation graph to delete consecutive SPLIT-MERGE edges. (a) A local region of a transformation graph, that includes consecutive SPLIT-MERGE edges. The numerical values stored in nodes P_1, P_2, C_1, C_2, C_3 must all be equal to the same real number v. (b) An edited but equivalent version of the region shown in (a). We note that nodes C_1, C_2 and C_3 have been deleted, P_1 is directly connected to what were the descendants of C_1 in (a), and P_2 is directly connected to what were the descendants of C_3 in (a). .

For example, in the graph of Figure 3.5, consider the move of the "3" node of the top layer to a "5", and the move of the "1" node of the top layer to a "7". The order of those two moves is interchangeable. On the other hand, the move of the "1" node to a "7" must occur before the move of the "7" to an "8".

3.3.2 The Monotonicity Lemma

Using transformation graphs, we can derive certain claims about transformations of a time series X into a time series Y. We will use these claims to derive an efficient algorithm for computing MSM distances.

We define two transformation graphs to be equivalent transformation graphs if they have the same input time series and output time series. Note that any transformation graph fully specifies an input time series X, an output time series Y, and a partially ordered set of operations that converts X into Y.

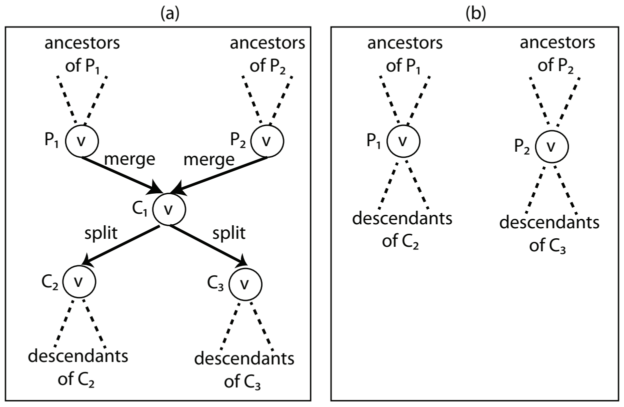


Figure 3.7. Editing a transformation graph to delete consecutive MERGE-SPLIT edges. (a) A local region of a transformation graph, that includes consecutive MERGE-SPLIT edges. The numerical values stored in nodes P_1, P_2, C_1, C_2, C_3 must all be equal to the same real number v. (b) An edited but equivalent version of the region shown in (a). We note that nodes C_1, C_2 and C_3 have been deleted, P_1 is directly connected to what were the descendants of C_2 in (a), and P_2 is directly connected to what were the descendants of C_3 in (a)..

Proposition 1. *Let G be a transformation graph that converts time series X into time series Y. If G includes any consecutive SPLIT-MERGE edges, we can convert G into an equivalent transformation graph G', such that G' is at least as cheap as G, and G' contains no consecutive SPLIT-MERGE edges.*

Proof: There are two possible local topologies corresponding to consecutive SPLIT-MERGE edges. The first is the case where the Merge operation directly undoes the effects of the preceding Split operation. In that case, clearly these two operations cancel each other out and can be deleted without changing the output of the transformation.

Figure 3.6 illustrates the local topology corresponding to the second case. In that figure, the numerical values stored in nodes P_1, P_2, C_1, C_2, C_3 are all equal to the same value v, because of the definition of the Split and Merge operations. The consecutive Split and Merge operations have the net effect of converting two consecutive v values (stored in nodes P_1 and P_2) into two consecutive v values (of nodes

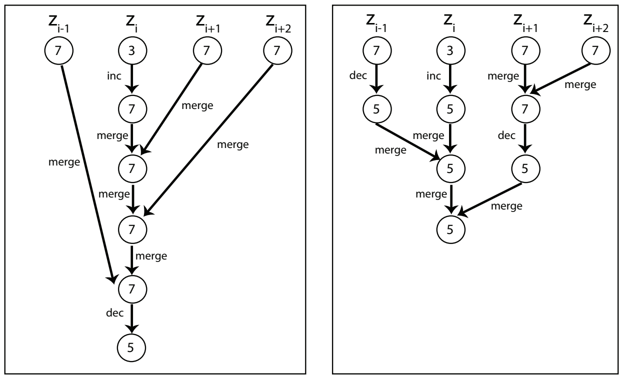


Figure 3.8. Left: a local region of a transformation graph that includes a non-monotonic path, of the form INC-MERGE-MERGE-MERGE-DEC. This region transforms series $(7, 3, 7, 7)$ into single-element series (5). The cost is $6 + 3c$. Right: The result of converting the region shown on the left into an equivalent but monotonic region, with the same cost $6 + 3c$, following the description of Case 1 in the proof of Proposition 3..

C_1 and C_3), and thus they can be deleted without changing the output of the graph. By deleting those two operations and editing the local topology as shown in the figure, we obtain an equivalent transformation graph, that is cheaper than the original transformation graph by a difference of $2c$. $\square$

Proposition 2. *Let G be a transformation graph that converts time series X into time series Y. If G includes any consecutive MERGE-SPLIT edges, we can convert G into an equivalent transformation graph G', such that G' is at least as cheap as G, and G' contains no consecutive MERGE-SPLIT edges.*

Proof: Figure 3.7 illustrates the local topology corresponding to consecutive MERGE-SPLIT edges. The Merge operation merges two values of v into one, and the Split operation directly undoes the effects of the preceding Merge operation, by recreating two values of v. Thus, we can delete both the Merge and the Split operation without changing the final output of the transformation graph. $\square$

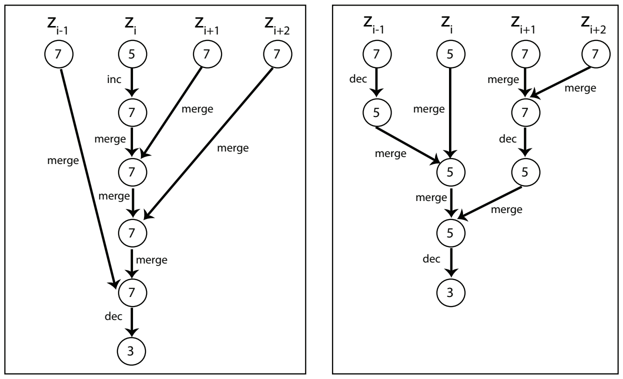


Figure 3.9. Left: a local region of a transformation graph that includes a non-monotonic path, of the form INC-MERGE-MERGE-MERGE-DEC. This region transforms series $(7, 5, 7, 7)$ into single-element series (3). The cost is $6 + 3c$. Right: The result of converting the region shown on the left into an equivalent but monotonic region, with the same cost $6 + 3c$, following the description of Case 2 in the proof of Proposition 3. We note that no INC edges appear in the region on the right..

We define a path of a transformation graph to be a monotonic path if it does not contain both INC and DEC edges. We define a monotonic transformation graph to be a transformation graph that only contains monotonic paths. We define a monotonic transformation to be a transformation whose transformation graph is monotonic.

Proposition 3. *Let X and Y be two time series. Let $\mathbb{S}$ be a transformation that converts X into Y. If $\mathbb{S}$ is not monotonic, we can convert $\mathbb{S}$ into another transformation $\mathbb{S}'$, that also converts X into Y, is as cheap or cheaper than $\mathbb{S}$, and is monotonic.*

Proof: This proposition has a long proof, because we have to cover several different cases. We can assume that transformation $\mathbb{S}$ has already been processed as described in Propositions 1 and 2, so that there are no consecutive SPLIT-MERGE edges in the transformation graph. Also, any consecutive INC-DEC edges or DEC-INC edges are clearly suboptimal, and can be replaced with a single INC or DEC edge. So, we can ignore such cases from here on.

If the transformation graph is not monotonic, it must have a non-monotonic path. Then, the path must have a subpath, whose one end is an INC edge, the other end is a DEC edge, and the intermediate edges are either all of type MERGE or all of type SPLIT (based on Propositions 1 and 2). We will primarily consider the case where the path is of the form INC-MERGE-...-MERGE-DEC, because once we prove the proposition for that case, the proof for the other cases is straightforward. Two examples of an INC-MERGE-...-MERGE-DEC path and its surrounding local topology are illustrated in Figures 3.8 and 3.9. We advise the reader to refer to these examples while reading the remainder of this proof.

Since we can re-order operations in $\mathbb{S}$ into any ordering compatible with the partial order imposed by the transformation graph, we choose to use an order in which the operations specified by the INC-MERGE-...-MERGE-DEC path are applied consecutively. Let $Z = (z_1, \ldots, z_t)$ be the time series to which the first operation of the path is applied. In that case, the INC edge corresponds to some operation $\text{Move}_{i,v}$, for some i and some positive v. This operation moves the i-th element of Z from value z_i to value $z_i + v$. Then, there is a sequence of Merge operations, that merge the i-th element with l elements $z_{i-l}, \ldots, z_{i-1}$, and with r elements $z_{i+1}, \ldots, z_{i+r}$, which all have the same value $z_i + v$. It is possible for either l or r to be equal to 0. In Figure 3.8, $l = 1$, $r = 2$, $z_i = 3$, $v = 4$, and $z_{i-1} = z_{i+1} = z_{i+2} = 7$.

After all the Merge operations have been applied, elements $z_{i-l}, \ldots, z_{i+r}$ have been converted into a single element, with value $z_i + v$. The final DEC edge corresponds to changing value $z_i + v$ to a new value $z_i + v - v'$, where v' is a positive real number ($v' = 2$ in Figure 3.8). The net result of all those operations is merging elements $z_{i-l}, \ldots, z_{i+r}$ of time series Z into a single value $z_i + v - v'$. The overall cost of all these operations is $v + v' + (l + r) * c$, since we do two Move operations of magnitude v and v' respectively, and $l + r$ Merge operations. Our task is now to show that we can convert all elements $z_{i-l}, \ldots, z_{i+r}$ into a single element with value $z_i + v - v'$, with less than or equal cost, and without having a non-monotonic path.

We will consider two cases: $v \geq v'$, and $v < v'$.

Case 1: $v \geq v'$. Figure 3.8 illustrates an example of this case. Consider replacing the sequence of operations specified by the INC-MERGE-…-MERGE-DEC path with the following combination of operations:

1. We move z_i up to $z_i + v - v'$, with cost $v - v'$.
2. If $l > 0$, we merge elements $z_{i-l}, \ldots, z_{i-1}$ into a single element whose value is $z_i + v$, and we move that single element down to $z_i + v - v'$. The cost of these operations is $(l - 1) * c + v'$.
3. If $r > 0$, we merge elements $z_{i+1}, \ldots, z_{i+r}$ into a single element whose value is $z_i + v$, and we move that single element down to $z_i + v - v'$. The cost of these operations is $(r - 1) * c + v'$.
4. We merge the results of steps 1, 2, and 3 into a single element. The cost here is at most 2 * c, it can be less if $l = 0$ or $r = 0$.

Step 4 must take place after steps 1, 2, and 3, whereas the order of steps 1, 2, and 3 is not important. Overall, the total cost of the above four steps is $(l + r) * c + v + v'$, which is equal to the cost of the original INC-MERGE-…-MERGE-DEC path. In the special case where $l = 0$ or $r = 0$, the total cost becomes $(l + r) * c + v$, which is better than the original cost. At the same time, the local topology resulting from these changes to the transformation graph includes only monotonic paths. Furthermore, the resulting transformation is at least as cheap as the original transformation. Figure 3.8 shows an example of this process, the local topology corresponding to the original INC-MERGE-…-MERGE-DEC, and the local topology corresponding to the new combination of operations.

Case 2: $v < v'$. Figure 3.9 illustrates an example of this case. Consider replacing the sequence of operations specified by the INC-MERGE-...-MERGE-DEC path with the following combination of operations:

1. If $l > 0$, we merge elements $z_{i-l}, \ldots, z_{i-1}$ into a single element whose value is $z_i + v$, and we move that single element down to value z_i. The cost of these operations is $(l - 1) * c + v$.
2. If $r > 0$, we merge elements $z_{i+1}, \ldots, z_{i+r}$ into a single element whose value is $z_i + v$, and we move that single element down to value z_i. The cost of these operations is $(r - 1) * c + v$.
3. We merge z_i and the results of steps 1 and 2 into a single element, with value z_i. The cost here is 2 * c, or less if $l = 0$ or $r = 0$.
4. We move the result of step 3 down to final value $z_i + v - v'$, with cost $v' - v$.

Steps 1 and 2 can take place in any order, but step 3 must be taken after steps 1 and 2, and step 4 after step 3. The cost of these four steps is at most $(l + r) * c + v + v'$, so it is not greater than the cost of the original sequence of operations. At the same time, the local topology resulting from these changes to the transformation graph includes only monotonic paths. Furthermore, the resulting transformation is at least as cheap as the original transformation. Figure 3.9 shows an example of this process, the local topology corresponding to the original INC-MERGE-...-MERGE-DEC, and the local topology corresponding to the new combination of operations.

We can now briefly consider the remaining cases of non-monotonic paths. The proof for paths of the form form DEC-MERGE-...-MERGE-INC is a direct adaptation of the proof we provided for paths of the form INC-MERGE-...-MERGE-DEC. For paths of the form INC-SPLIT-...-SPLIT-DEC or DEC-SPLIT-...-SPLIT-INC, we use the fact that, as discussed in Section 3.1 (when demonstrating that MSM is symmetric), any transformation of X into Y can be inverted, to produce an equal-cost transformation of Y into X. Thus, if, for some transformation $\mathbb{S}$ of X into Y, the corresponding transformation graph contains a path of the form INC-SPLIT-...-SPLIT-DEC or DEC-SPLIT-...-SPLIT-INC, then for the inverse transformation $\mathbb{S}^{-1}$ of Y into X the transformation graph contains a path of the form INC-MERGE-...-MERGE-DEC or DEC-MERGE-...-MERGE-INC. We can edit $\mathbb{S}^{-1}$ to remove such paths, and then invert it again, to obtain a transformation that changes X into Y and that does not include paths of the form INC-SPLIT-...-SPLIT-DEC or DEC-SPLIT-...-SPLIT-INC.

At this point, we have shown that, for any type of non-monotonic path in a transformation graph, we can edit the graph so that the non-monotonic path is replaced with an arrangement of monotonic paths, and we have shown that the edited graph is equivalent to G and at least as cheap as G. By repeating such edits, we can convert any transformation graph G into an equivalent, monotonic, and at least as cheap transformation graph G', and thus we have concluded the proof of Proposition 3. $\square$

We are now ready to state and prove the monotonicity lemma, which is a key lemma for describing, in Section 3.4, the algorithm for computing MSM distances.

Proposition 4. *(Monotonicity lemma) For any two time series X and Y, there exists an optimal transformation that converts X into Y and that is monotonic.*

Proof: Let $\mathbb{S}$ be an optimal transformation that converts X into Y. Let G be the transformation graph corresponding to applying $\mathbb{S}$ to X. If G is not monotonic, we can convert G to a monotonic graph G' that is at least as cheap as G (and thus also optimal), by editing G as described in the proofs of Propositions 1, 2, and 3. Then, any transformation S compatible with G' is an optimal and monotonic transformation of X into Y. $\square$

3.4 Computing the MSM Distance

Let $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ be two time series. Figure 3.10 describes a simple dynamic programming algorithm for computing the MSM distance between X and Y. For each (i, j) such that $1 \leq i \leq m$ and $1 \leq j \leq n$, we define $\text{Cost}(i, j)$ to be the MSM distance between the first i elements of X and the first j elements of Y. This way, the MSM distance between X and Y is simply $\text{Cost}(m, n)$.

As the algorithm on Figure 3.10 shows, for $i > 1$ and $j > 1$, $\text{Cost}(i, j)$ can be computed recursively based on $\text{Cost}(i, j-1)$, $\text{Cost}(i-1, j)$, and $\text{Cost}(i-1, j-1)$. In this section we explain why it is correct to define the Cost function in this recursive manner, and we fully specify how to actually compute the Cost function.

First, we note that $\text{Cost}(1, 1)$ is simply the cost of moving x_1 to y_1, so this is a trivial case. The interesting case is when $i > 1$ or $j > 1$. In that case, we know from the monotonicity lemma that there exists an optimal monotonic transformation $\mathbb{S}_{i,j}$ converting $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_j)$. We use notation $G_{i,j}$ for the transformation graph corresponding to applying $\mathbb{S}_{i,j}$ to X. In $G_{i,j}$ there is a monotonic path moving x_i to y_j. There can be three cases for that path, that we need to analyze separately.

Case 1 (applicable if $i > 1$ and $j > 1$): the monotonic path taking x_i to y_j does not include any SPLIT or MERGE edges. In that case, without loss of generality, we can assume that the monotonic path taking x_i to y_j contains a single INC or DEC edge. We refer the reader to Figure 3.11 for an example.

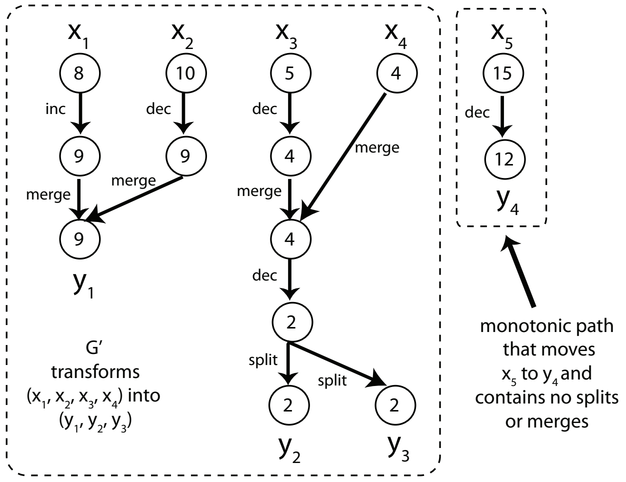
Consider the transformation graph G' that we obtain by removing the INC or DEC edge connecting x_i to y_j from transformation graph $G_{i,j}$. We show by contradic-

```
function MSM_Distance($X$, $Y$)

Inputs:

  Time series $X = (x_1, \ldots, x_m)$
  Time series $Y = (y_1, \ldots, y_n)$

Initialization:
  $\text{Cost}(1,1) = |x_1 - y_1|$.
  For $i = 2, \ldots, m$:
    $\text{Cost}(i,1) = \text{Cost}(i-1,1) + C(x_i, x_{i-1}, y_1)$
  For $j = 2, \ldots, n$:
    $\text{Cost}(1,j) = \text{Cost}(1,j-1) + C(y_j, x_1, y_{j-1})$

Main Loop:
  For $i = 2, \ldots, m$:
    For $j = 2, \ldots, n$:
      $\text{Cost}(i,j) = \min\{\text{Cost}(i-1,j-1) + |x_i - y_j|,$
                       $\text{Cost}(i-1,j) + C(x_i, x_{i-1}, y_j),$
                       $\text{Cost}(i,j-1) + C(y_j, x_i, y_{j-1})\}$

Output: The MSM distance $D(X,Y)$ is $\text{Cost}(m,n)$.
```

Figure 3.10. A simple, quadratic-time algorithm for computing the MSM distance between two time series $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$. Function C, used in computing values for the Cost array, is defined in Equation 3.9..

tion that G' defines an optimal transformation of $(x_1, \ldots, x_{i-1})$ into $(y_1, \ldots, y_{j-1})$. If G' is not optimal, then there exists an optimal transformation $\mathbb{S}_1$ that has a smaller cost than G'. If we add a Move operation to the end of $\mathbb{S}_1$, that moves x_i to y_j, we obtain a transformation that converts $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_j)$ and that is cheaper than $\mathbb{S}_{i,j}$, which was assumed to be optimal. Therefore, we have reached a contradiction.

Consequently, if Case 1 holds, we obtain an optimal transformation $\mathbb{S}_{i,j}$ by adding a move operation (moving x_i to y_j) to an optimal transformation converting



x1
x2
x3
x4
x5
8
10
5
4
15
inc
dec
dec
merge
dec
9
9
4
12
merge
merge
merge
y4
9
4
y1
dec
2
monotonic path that moves x5 to y4 and contains no splits or merges
G′ transforms (x1, x2, x3, x4) into (y1, y2, y3)
split
split
2
2
y2
y3

Figure 3.11. An example of Case 1 for an optimal monotonic transformation graph $G_{i,j}$. $G_{i,j}$ maps $(x_1, \ldots, x_i)$ to $(y_1, \ldots, y_j)$. In Case 1, $G_{i,j}$ is obtained from an optimal transformation graph G' mapping $(x_1, \ldots, x_{i-1})$ to $(y_1, \ldots, y_{j-1})$, by adding to G' a Move operation that moves x_i to y_j. In the example shown here, $i = 5$ and $j = 4$. .

$(x_1, \ldots, x_{i-1})$ into $(y_1, \ldots, y_{j-1})$. It follows readily that, if Case 1 holds, $\text{Cost}(i, j) = \text{Cost}(i-1, j-1) + |x_i - y_j|$.

Case 2 (applicable if $i > 1$): in the monotonic path moving x_i to y_j, the first non-move operation is a Merge. In the transformation graph $G_{i,j}$, that first Merge operation creates a node M with two parents. One of those parents, that we call P_i, has x_i as an ancestor. The other parent, that we call P_{i-1}, has x_{i-1} as an ancestor. There is a path passing through P_{i-1} and M that connects x_{i-1} to y_j. There is another path passing through P_i and M that connects x_i to y_j. Since the transformation is monotonic, the value v stored at node M must be between x_{i-1} and y_j, and also between x_i and y_j. Figure 3.12 illustrates three examples, with the position of node M indicated.

For Case 2, there are three subcases that we need to address. An example for each subcase is shown in Figure 3.12.

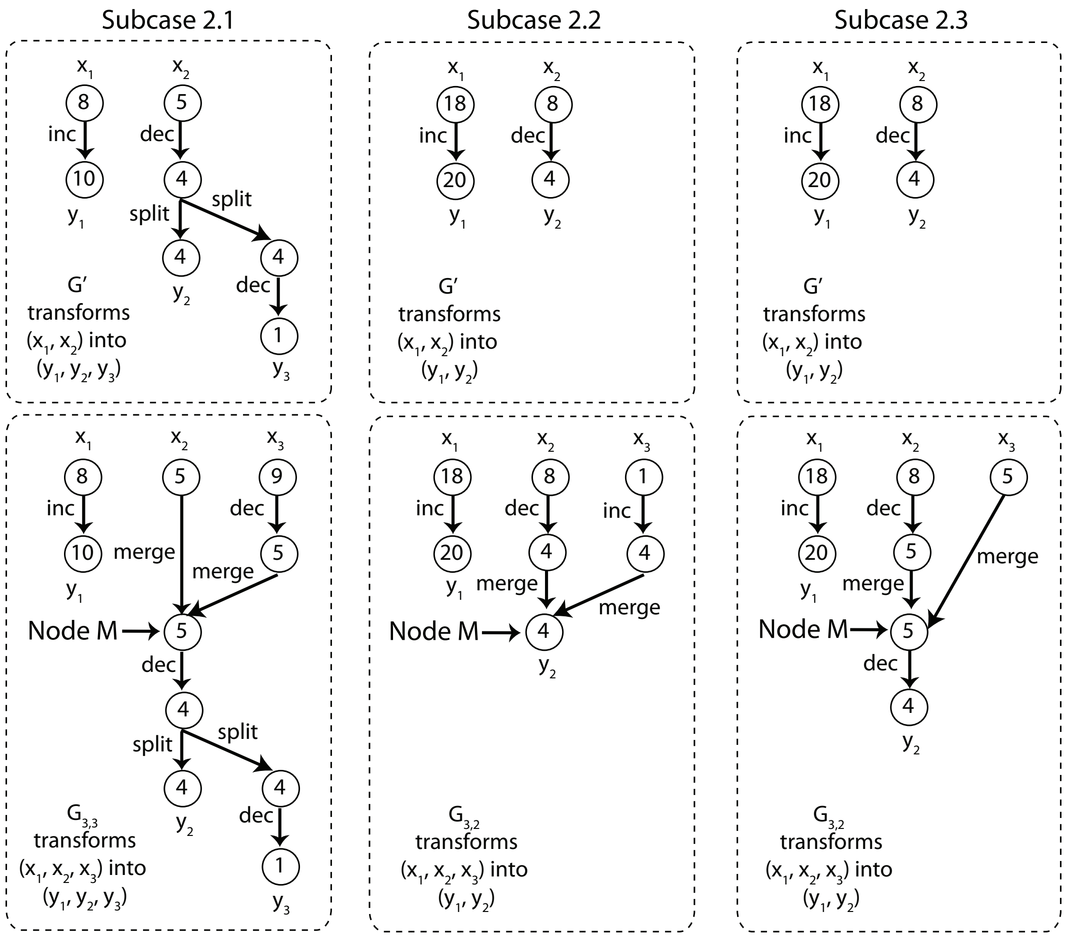

Subcase 2.1
Subcase 2.2
Subcase 2.3
G′ transforms (x1, x2) into (y1, y2, y3)
G′ transforms (x1, x2) into (y1, y2)
G3,3 transforms (x1, x2, x3) into (y1, y2, y3)
G3,2 transforms (x1, x2, x3) into (y1, y2)
Node M
inc
dec
split
merge

Figure 3.12. Examples of the three subcases of Case 2 for an optimal monotonic transformation graph $G_{i,j}$. $G_{i,j}$ maps $(x_1, \ldots, x_i)$ to $(y_1, \ldots, y_j)$. In Case 2, $G_{i,j}$ is obtained from an optimal transformation graph G' mapping $(x_1, \ldots, x_{i-1})$ to $(y_1, \ldots, y_j)$. Subcase 2.1: the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to x_{i-1} than to y_j. In the example for Subcase 2.1, $i = 3$ and $j = 3$. In Subcase 2.2, the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to y_j than to x_{i-1}. In the example for Subcase 2.2, $i = 3$ and $j = 2$. In Subcase 2.3, the value of x_i is between the value of x_{i-1} and the value of y_j. In the example for Subcase 2.3, $i = 3$ and $j = 2$. Note that, in this example, in the optimal transformation from (x_1, x_2) to (y_1, y_2), x_2 moves directly from value 8 to value 4. In the optimal transformation from (x_1, x_2, x_3) to (y_1, y_2), x_2 moves first to an intermediate value of 5, that allows a merge with x_3, and then to value 4..

- Subcase 2.1: the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to x_{i-1} than to y_j. Then, x_i first moves to value x_{i-1}, and then merges.

- Subcase 2.2: the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to y_j than to x_{i-1}. Then, x_i first moves to value y_j, and then merges.
- Subcase 2.3: the value of x_i is between the value of x_{i-1} and the value of y_j. In that case, x_i merges immediately with a value along the monotonic path that moves x_{i-1} to y_j.

In all three subcases, by removing the one or two operations linking x_i with node M from the transformation graph $G_{i,j}$, we obtain a transformation graph G' that converts $(x_1, \ldots, x_{i-1})$ into $(y_1, \ldots, y_j)$. As in Case 1, we can show that if G' is suboptimal, then $G_{i,j}$ is suboptimal (which is a contradiction). Consequently, G' is optimal, and if Case 2 holds then $\text{Cost}(i, j) = \text{Cost}(i-1, j) + C(x_i, x_{i-1}, y_j)$, where $C(x_i, x_{i-1}, y_j)$ is defined as follows:

$$C(x_i, x_{i-1}, y_j) = \begin{cases} c \ \text{ if } x_{i-1} \leq x_i \leq y_j \text{ or } x_{i-1} \geq x_i \geq y_j \\ c + \min(|x_i - x_{i-1}|, |x_i - y_j|) \text{ otherwise} \end{cases} \tag{3.9}$$

In Figure 3.12, for Subcase 2.3 in particular, we should note that the transformation graph obtained by removing the Merge operation from the bottom graph is *not* identical to the top graph. However, both graphs have equal cost. The only difference is that in the top graph x_{i-1} moves directly from a value of 8 to a value of 4, and in the bottom graph x_{i-1} moves first to an intermediate value of 5, and then to the final value of 4.

Case 3 (applicable if $j > 1$): in the monotonic path moving x_i to y_j, the first non-move operation is a Split. We omit the details here, but the analysis for this case is a direct adaptation of the analysis for Case 2. In summary, in Case 3 we can obtain from transformation graph $G_{i,j}$ an optimal transformation graph G' that converts $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_{j-1})$, so that $\text{Cost}(i, j) = \text{Cost}(i, j-1) + C(y_j, x_i, y_{j-1})$.

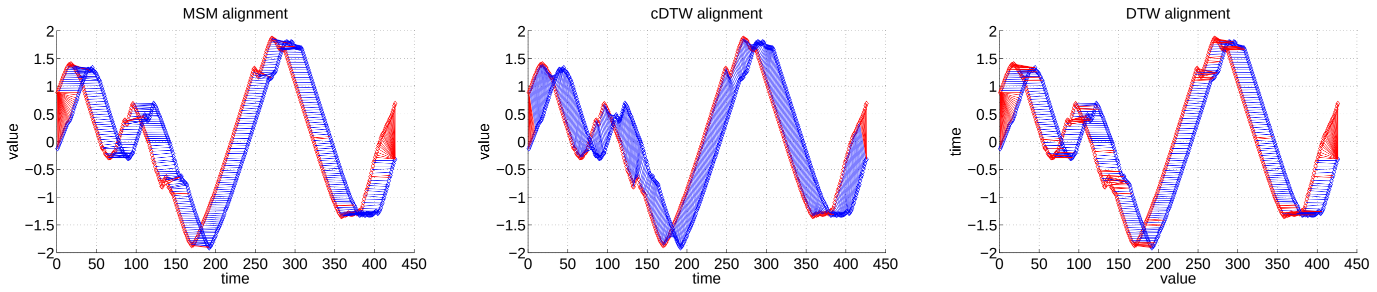


Figure 3.13. An example, from the Yoga dataset, of a query that MSM classifies correctly whereas cDTW and DTW classify incorrectly, due to time shift. The query series is shown in blue. Its nearest neighbor according to MSM (which belongs to the same class) is shown in red. The alignments computed by MSM (left), cDTW (middle), and DTW (right) are shown via links connecting corresponding elements. .

Based on the above considerations, the algorithm on Figure 3.10 checks which of the three cases leads to a cheaper transformation of $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_j)$. The cost of the transformation corresponding to each case is computed in a few operations, using the already computed values for $\text{Cost}(i, j-1)$, $\text{Cost}(i-1, j)$, and $\text{Cost}(i-1, j-1)$. The algorithm for computing MSM distances is fairly simple, and can be implemented in a few lines of code.

Computing $\text{Cost}(i, j)$ takes constant time for each (i, j). Therefore, the time complexity of computing the MSM distance is $O(mn)$. The $O(mn)$ complexity is the same as the time complexity of DTW (without the diagonality constraint [35]) and ERP. The Euclidean distance, in contrast, has linear time complexity $O(m)$, and $n = m$ in that case. Constrained DTW [35], that utilizes the diagonality constraint, also has linear time complexity if we consider that the radius around the diagonal does not depend on the length of the time series.

3.5 Experiments

We compare MSM to cDTW, DTW, ERP, and the Euclidean distance, based on the 1-nearest neighbor classification error rate attained on the 20 time series datasets available on the UCR time series archive [3]. We should note that, while the UCR time series website shows results on 22 datasets, only 20 of those datasets are publicly available, and those are the 20 datasets that we have used. A note on the website indicates that two of those datasets (namely, the "Car" and "Plane" datasets) are still not publicly available.

The MSM algorithm has one free parameter, namely c, the cost of every Split and Merge operation. For each of the 20 datasets, the value for c was chosen from the set $\{0.01, 0.1, 1, 10, 100\}$, using leave-one-out cross-validation on the training set. It is important to emphasize that c was *not* optimized based on results on the test data. Overall we have found it fairly straightforward to pick a value for c by simply trying those five values on the training data.

We should note that considering a lot of possible values for c could slow down the training phase significantly, as a separate cross-validation measurement must be obtained for each individual value. In our experiments, MSM produced competitive error rates while considering only five widely-spaced values (differing by factors of 10) for c. Considering only five widely-spaced values demonstrates that no careful fine tuning of c was needed to obtain good results.

Tables 3.2 shows the error rate for each method on each dataset and the statistical significance (p-value) of the results. The p-value specifically measures the statistical significance of the difference between the top two methods for each data set.

Table 3.1 shows characteristics of each dataset, the parameter values used by MSM and cDTW for that dataset.

We note that for each method there are some datasets where that method is at least as accurate as the other four methods. MSM produces lower error rates than its competitors in 10 datasets. Each of DTW and ERP produces the lowest error rate in two datasets. In the remaining six datasets, two or more methods tie for lowest error rate. Table 3.3 shows, for each competitor of MSM, the number of datasets where MSM produces respectively better accuracy, equal accuracy, and worse accuracy compared to the competitor.

Our primary goal in these experiments has been to demonstrate that MSM has competitive performance on 1-nearest neighbor classification, compared to cDTW, DTW, and ERP. We are *not* making a claim that MSM is a fundamentally more accurate measure than cDTW, DTW, or ERP. Our interpretation of the results is that all these methods are valuable, and any one of them may outperform the other methods in a new dataset. At the same time, MSM has some attractive theoretical properties that DTW or ERP do not have.

A natural question to ask is how to determine which of these methods to use on a new dataset. A simple answer to that question is to simply evaluate all methods on the training set (using leave-one-out cross-validation), and choose the method with the lowest error rate. We have tried that approach, and we show the results on the rightmost two columns of Table 3.2. If two or more methods tied on the training set, we show the average test error of those methods. We tried two variants: in the CV+MSM variant, we chose for each dataset the best out of all five methods. In the CV-MSM variant we excluded MSM from consideration.

In those results, CV+MSM matched the best error rate in 12 datasets and CV-MSM matched the best error rate (excluding MSM) in 11 datasets. In head-to-head comparison with each of the individual methods they included, both CV+MSM and CV-MSM gave better results in more datasets than they gave worse results. Both CV+MSM and CV-MSM had lower average error rates than any of the individual methods that they included. Thus, these results demonstrate that cross-validation is a good way to choose automatically which method to use in each dataset. Furthermore, we note that CV+MSM had a lower error rate than CV-MSM in 10 datasets, and higher error rate in only three datasets. This result further illustrates the advantages of considering MSM as an alternative to DTW and ERP in practical applications.

In Figures 3.13, 3.14 and 3.15 we illustrate some specific examples where MSM gives better or worse accuracy compared to its competitors. These examples help build some intuition about how the behavior of different methods can influence classification results.

Figure 3.13 shows an example where MSM classifies the query correctly, whereas cDTW and DTW give the wrong answer. The main difference between the query and its MSM-based nearest neighbor is time shift, which causes mismatches at the beginning and the end of the sequences. MSM erases (via small moves and merges) the mismatched points with relatively low cost. In DTW, the cost of matching the extra points prevents this training object from being the nearest neighbor of the query. The time shift affects cDTW even more severely, as the warping window is too small to compensate for the shift.

Figure 3.14 shows another example where the query is classified correctly by MSM, and incorrectly by cDTW and DTW. Here, the query contains a valley between times 80 and 100, and that valley is not matched well by the query's MSM-based nearest neighbor. MSM "collapses" the mismatched valley to a single point with relatively low cost. In DTW, the cost of matching elements of the training object to points in that valley is large enough to prevent this training object from being the nearest neighbor of the query.

Figure 3.15 shows a case where MSM gives the wrong answer, whereas cDTW, DTW and ERP give the right answer. For that query, we show both its MSM-based nearest neighbor (denoted as D), which belongs to the wrong class, as well as its MSM-based nearest neighbor (denoted as S) among training examples of the same class as the query. The main difference between the query and D is a peak and a valley that the query exhibits between time 200 and time 250. This difference gets penalized by DTW, cDTW, and ERP, and thus, according to those measures the query is closer to S than to D. On the other hand, the MSM distance between the query and D is not affected much by the extra peak and valley of the query. Thus, according to MSM, the query is closer to D than to S.

Figures 3.14 and 3.15 indicate that MSM penalizes extra peaks and valleys less severely than cDTW, DTW, and ERP. This may be a desirable property in data where such extra peaks and valleys appear due to outlier observations. We simulated this situation in the following experiment: for each test example of each of the 20 UCR datasets, we modified that example by adding an extra peak. The width of the peak was 10 elements, and the height of the peak was chosen randomly and uniformly between 0 and 80. Two examples of this modification are shown on Figure 3.16. We measured the error rates of MSM and its competitors on this modified dataset. We note that the training examples were not modified, and thus the free parameters chosen via cross-validation for MSM and cDTW remained the same.

Due to lack of space, the table of error rates for this experiment is provided as supplementary material. The summary of those results is that, while the average error rates of all methods increase, MSM suffers significantly less than its competitors. MSM gives lower error rate than cDTW, DTW, and the Euclidean distance on all 20 datasets. Compared to ERP, MSM does better on 16 datasets, worse in 3 datasets, and ties ERP in 1 dataset.

Finally, Table 3.4 compares the efficiency of MSM to that of its competitors. As expected, the Euclidean distance and cDTW are significantly faster than MSM, DTW, and ERP. In all datasets the running time for MSM was between 0.97 and 2 times the running time of DTW and ERP. Running times were measured on a PC with 64-bit Windows 7, an Intel Xeon CPU running at 2GHz, 4GB of RAM, and using a single-threaded implementation.

3.6 Discussion

This chapter has described MSM, a novel metric for time series, that is based on the cost of transforming one time series into another using a sequence of individual Move, Split, and Merge operations. MSM has the attractive property of being both metric and invariant to the choice of origin, whereas DTW is not metric, and ERP is not invariant to the choice of origin. These properties may make MSM a more appealing choice, compared to existing alternatives, in various domains. Metricity, in particular, allows the use of a large number of existing tools for indexing, clustering and visualization, that have been designed to work in arbitrary metric spaces.

We have presented a quadratic-time algorithm for computing the MSM distance between two time series. A large part of the paper has been dedicated to explaining the algorithm and proving its correctness. At the same time, despite the relatively complex proof, the actual algorithm is quite short and easy to implement, as shown on Figure 3.10, and on the implementations we have posted online.

Experiments on all 20 datasets available at the UCR time series archive [3] demonstrate that, in ten of the 20 datasets, MSM produces lower nearest neighbor classification error rate than constrained DTW, unconstrained DTW, ERP, and the Euclidean distance. The fact that MSM gave the best accuracy in several datasets supports the conclusion that MSM is a method worth being aware of and experimenting with, in domains where practitioners currently use DTW or ERP. The attractive theoretical properties of MSM are an additional factor that can make MSM an appealing choice, compared to existing alternatives.

Table 3.1. Information on the 20 UCR datasets that we have used in these experiments. For each dataset, the table shows the number of classes, the number of training objects, the number of test objects, the length of each sequence in the dataset, the value of c used by MSM on that dataset, and the length of the warping window (as specified in [3]) used by cDTW on that dataset.

Dataset	class num.	train. size	test size	seq. length	MSM c	cDTW param.
Coffee	2	28	28	286	0.01	3
CBF	3	30	900	128	0.1	11
ECG	2	100	100	96	1	0
Synthetic	6	300	300	60	0.1	6
Gun Point	2	50	150	150	0.01	0
FaceFour	4	24	88	350	1	2
Lightning-7	7	70	73	319	1	5
Trace	4	100	100	275	0.01	3
Adiac	37	390	391	176	1	3
Beef	5	30	30	30	0.1	0
Lightning-2	2	60	61	637	0.01	6
OliveOil	4	30	30	570	0.01	1
OSU Leaf	6	200	242	427	0.1	7
SwedishLeaf	15	500	625	128	1	2
Fish	7	175	175	463	0.1	4
FaceAll	14	560	1690	131	1	3
50words	50	450	455	270	1	6
Two Patterns	4	1000	4000	128	1	4
Wafer	2	1000	6174	152	1	1
Yoga	2	300	3000	426	0.1	2
average						

Table 3.2. 1-nearest neighbor classification error rates attained by MSM, constrained DTW (denoted as cDTW), unconstrained DTW (denoted as DTW), ERP, and the Euclidean distance, on each of the 20 datasets in the UCR repository of time series datasets [3]. The last row indicates the average error rate over all 20 datasets. We also show, for each dataset, the statistical significance(p-value) of the difference between the two best-performing methods for that dataset. The last two columns show the results of the CV+MSM and CV-MSM hybrid methods, described in the text, where the distance measure used for each dataset is the one that minimizes training error.

Dataset	MSM	cDTW	DTW	ERP	Euclid	p value	CV +MSM	CV -MSM
Coffee	0.236	0.179	0.179	0.25	0.25	0.5	0.179	0.179
CBF	0.012	0.004	0.003	0.003	0.148	0.5	0.006	0.003
ECG	0.11	0.12	0.23	0.13	0.12	0.3285	0.117	0.12
Synthetic	0.027	0.017	0.007	0.037	0.12	0.2076	0.007	0.007
Gun Point	0.06	0.087	0.093	0.04	0.087	0.1595	0.078	0.087
FaceFour	0.057	0.114	0.17	0.102	0.216	0.0224	0.057	0.114
Lightning-7	0.233	0.288	0.274	0.301	0.425	0.2476	0.288	0.288
Trace	0.07	0.01	0	0.17	0.24	0.1599	0	0
Adiac	0.384	0.391	0.396	0.379	0.389	0.3276	0.384	0.379
Beef	0.5	0.467	0.5	0.5	0.467	0.5	0.467	0.467
Lightning-2	0.164	0.131	0.131	0.148	0.246	0.5	0.131	0.131
OliveOil	0.167	0.167	0.133	0.167	0.133	0.5	0.167	0.167
OSU Leaf	0.198	0.384	0.409	0.397	0.483	< 0.0001	0.198	0.384
SwedishLeaf	0.104	0.157	0.21	0.12	0.213	0.0703	0.104	0.157
Fish	0.08	0.16	0.167	0.12	0.217	0.0448	0.08	0.16
FaceAll	0.189	0.192	0.192	0.202	0.286	0.2243	0.189	0.197
50words	0.196	0.242	0.31	0.281	0.369	0.01	0.196	0.242
Two Patterns	0.001	0.0015	0	0	0.09	0.5	0.0003	0
Wafer	0.004	0.005	0.02	0.011	0.005	0.2249	0.008	0.011
Yoga	0.143	0.155	0.164	0.147	0.17	0.2207	0.143	0.155
average	0.147	0.164	0.179	0.175	0.234		0.140	0.162

Table 3.3. We indicate the number of UCR datasets for which MSM produced better, equal, or worse accuracy compared to ERP, and also compared to DTW.

	MSM better	Tie	MSM worse
MSM vs. cDTW	13	1	6
MSM vs. DTW	12	1	7
MSM vs. ERP	13	2	5
MSM vs. Euclidean	18	0	2

Table 3.4. Runtime efficiency comparisons. For each dataset, in the MSM time column, the time it took in seconds to compute *all* distances from the entire test set to the entire training set. In the rightmost four columns we show the factor by which MSM was *slower* than each of cDTW, DTW, ERP, and the Euclidean distance.

Dataset	MSM time (sec)	cDTW factor	DTW factor	ERP factor	Euclidean factor
Coffee	1.59	13.25	1.49	1.42	159
CBF	16.05	3.77	1.51	1.61	94.41
ECG	2.88	3.56	1.55	1.46	48
Synthetic	10.89	8.71	1.51	1.35	36.3
Gun Point	4.14	10.89	1.48	1.18	103.5
FaceFour	9.59	17.76	1.99	0.97	479.5
Lightning-7	14.17	14.61	1.56	1.15	472.33
Trace	30.89	1.98	1.38	1.74	514.83
Adiac	103.33	2.52	1.28	1.03	178.16
Beef	6.11	2.54	1.14	0.97	611
Lightning-2	51.62	2.76	1.28	1.23	1720.67
OliveOil	8.83	1.89	1.04	1.03	883
OSU Leaf	259.16	2.94	1.27	1.09	959.85
SwedishLeaf	125.31	10.55	1.38	1.16	113.92
Fish	183.3	2.28	1.17	1.06	1018.33
FaceAll	491.56	12.89	1.25	1.28	111.46
50words	323.13	3.13	1.3	1.07	359.03
Two Patterns	2348.1	9.33	1.6	1.56	157.91
Wafer	3281.24	11.05	1.51	1.18	148
Yoga	4606.48	2.43	1.25	1.07	988.52
min		1.890	1.040	0.970	36.300
max		17.760	1.990	1.740	1,720.670
median		3.665	1.380	1.170	268.595
average		6.942	1.397	1.231	457.886

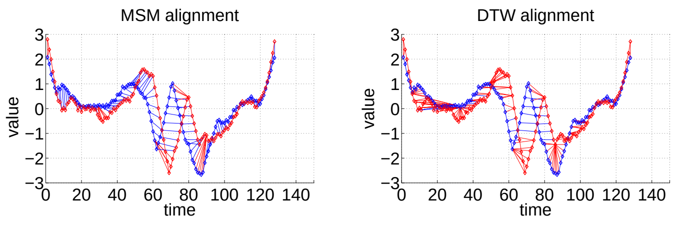


Figure 3.14. An example from the Swedish Leaf dataset, where MSM does better than DTW. The query series is shown in blue. Its nearest neighbor according to MSM is shown in red, and belongs to the same class as the query. For MSM (left) and and DTW (right), the alignment between the red and the blue series is shown via links connecting corresponding elements. .

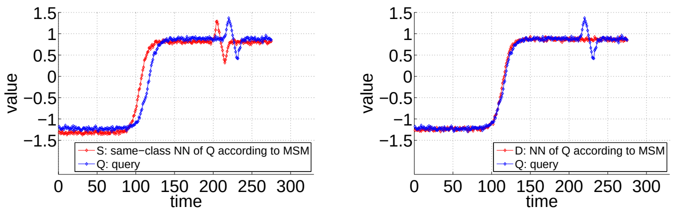


Figure 3.15. An example from the Trace dataset where MSM does worse than DTW and ERP. On the left, we show in blue Q, a query series, and in red S, the nearest neighbor (according to MSM) of Q among training examples of the same class as Q. On the right, we show in blue the same query Q, and in red we show D, the overall nearest neighbor (according to MSM), which belongs to a different class. .

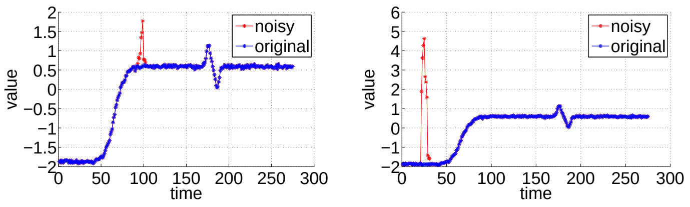


Figure 3.16. Two examples of a peak added to time series. In blue we show the original time series. The modified version is the same as the original time series, except for a small region (shown in red) of 10 values, where we have added a peak. .

CHAPTER 4

REDUCING JOINTBOOST-BASED MULTICLASS CLASSIFICATION TO PROXIMITY SEARCH

Many real-world applications involve recognizing a very large number of classes, a number that can range from thousands to millions. Examples of such applications include biometrics-based identification (based on faces and/or fingerprints), hand and human body pose classification, speech and sign language recognition, and generic object recognition using computer vision. An important problem in such domains is designing recognition methods that are scalable and that achieve efficient runtime in the presence of such a large number of classes.

Large margin methods, such as boosting [52, 53] and support vector machines (SVMs) [54], have been very successful in recent years in various pattern recognition domains. A common way to apply such methods to multiclass problems is to train a one-versus-all (OVA) classifier for each class [55, 56]. However, a major bottleneck of such approaches is that, given a new pattern to classify, all OVA classifiers must be applied to that pattern, so as to identify the OVA classifier that yields the strongest response. This leads to time complexity that is linear to the number of classes, which can lead to prohibitively large classification times in large multiclass domains with thousands or millions of classes.

JointBoost [56] is a method that has recently attracted significant attention in the vision community. In JointBoost, the OVA classifiers are trained jointly, and are forced to share features. In practice, this typically leads to both higher accuracy and faster classification time. Higher accuracy is obtained because the impact of each feature is evaluated simultaneously on multiple OVA problems, thus making the estimate of that impact more reliable than if measured only on a single OVA problem. Faster classification time is obtained because the total number of unique features that need to be extracted from an input image is drastically reduced, as features are shared among multiple classifiers.

Although JointBoost drastically improves feature extraction time, the time complexity of classifying an input image with JointBoost is still linear to the number of classes, as is the case with other OVA methods based on boosting or SVMs. As the number of classes becomes large, feature extraction time becomes a negligible part of total classification time, and most of the time is spent on computing the response of each OVA classifier. A key contribution of this thesis, as described in this chapter, is showing that, given a pattern to classify using JointBoost, identifying the strongest-responding OVA classifier for that pattern can be treated as a proximity search problem, and more specifically as a nearest neighbor search problem in a vector space. This result allows us to use a vast array of vector indexing methods, e.g., [57, 58, 41, 59, 60], so as to improve classification time.

To demonstrate the computational advantage that can be obtained by reducing JointBoost classification to nearest neighbor search, we have implemented and evaluated a simple, easy-to-use vector indexing method based on principal component analysis (PCA). In our experiments, the proposed method achieves a speedup of two orders of magnitude over standard JointBoost classification, in a hand pose recognition system where the number of classes is close to 50,000, with negligible loss in classification accuracy. Our method also yields promising results in experiments on the widely used FRGC-2 face recognition dataset, where the number of classes is 535.

4.1 Related Work

Large margin methods, such as boosting methods [52, 53] and support vector machines (SVMs) [54], have been widely used in recent years. Large margin methods are appealing because of their good generalization properties and their state-of-the-art performance in many applications (e.g., [56]). The standard strategy for applying large margin methods to a multiclass problem is to decompose the multiclass problem into a set of binary problems [55, 56].

Different types of multiclass-to-binary decompositions can be defined using error-correcting output codes [55, 61]. The most commonly used decompositions are into all-pairs problems, where a classifier is trained to discriminate between each pair of classes, or into one-vs.-all (OVA) problems, where, for each class, an OVA classifier is trained to discriminate between that class and all other classes. To classify a query, typically all binary classifiers are applied on the query pattern. An exception is the the DAGSVM method [62], that uses the all-pairs scheme but requires a number of classifier evaluations that is linear, not quadratic, to the number of classes.

One way to achieve classification time sublinear to the number of classes is to decompose the multiclass problem into a sublinear number of binary problems. In theory, recognizing n classes can be decomposed to $\log_2 n$ binary problems. However, such sublinear decompositions are rarely used because they define binary problems with unnatural and hard-to-learn class boundaries, leading to low classification accuracy. OVA and all-pairs decompositions, on the other hand, lead to more natural binary classification boundaries, and this explains the popularity of those decompositions in practice.

While OVA and all-pairs methods are frequently used in practice [63, 64, 56], the time complexity of those methods is at least linear to the number of classes. Linear complexity means that these methods are hard to scale to problems with a very large number of classes. Torralba, et al. [56] propose the JointBoost method for speeding up classification time. In JointBoost, the OVA models share weak classifiers among them. While sharing weak classifiers has improved both accuracy and efficiency in the experiments of [56], in JointBoost it is still the case that all OVA classifiers are applied to each pattern at runtime. Our method can be applied on top of JointBoost and significantly reduce classification time, as shown in the experiments.

In ClassMap [63], OVA classifiers and patterns are embedded into a common vector space, where the strongest responding OVA classifier for each pattern can be found more efficiently. ClassMap can be applied on top of more general large-margin methods, whereas the method proposed in this paper is designed for JointBoost-based OVA classifiers. On the other hand, the mapping proposed in this paper is lossless, and preserves information as to which OVA classifier gives the strongest response for a pattern; ClassMap does not guarantee preserving such information.

Some additional methods have been proposed for speeding up specific large multiclass problems. Efficient articulated pose estimation is achieved in [65] by combining hierarchical classifiers into a tree structure. Hierarchical template matching has been used for pedestrian detection [66] and articulated pose estimation [67]. Articulated pose can also be treated as a multidimensional regression problem, and estimators can be trained that directly map observations into vectors from a continuous pose space [68, 69]. However, many domains (e.g., face recognition) do not lend themselves readily either to hierarchical decomposition or to regression-based estimation. In contrast, our method can readily be applied in any domain where JointBoost is applicable, and thus is significantly more general than the above-mentioned domain-specific approaches.

4.2 Review: Multiclass Recognition Using JointBoost

Let $\mathbb{X}$ be a space of patterns, and $\mathbb{Y}$ be a finite set of class labels. Every pattern $X \in \mathbb{X}$ has a class label $L(X) \in \mathbb{Y}$. In JointBoost [56], for each class $y \in \mathbb{Y}$ a boosted classifier $H_y : \mathbb{X} \to \mathbb{R}$ is trained to discriminate between patterns of class y and all other patterns. Classifier H_y is of the following form:

$$H_y = \sum_{m=1}^{d} \alpha_{y,m} h_m + k_y \ , \tag{4.1}$$

where each h_m is a weak classifier with weight $\alpha_{y,m}$, and k_y is a class-specific constant that gives a way to encode a prior bias for each class y [56]. We should also note that, in JointBoost, weights $\alpha_{y,m}$ are constrained to be either 1 or 0 (depending on whether H_y is using weak classifier h_m or not), but the method proposed in this paper does not use that constraint, and can be applied regardless of the possible values for $\alpha_{y,m}$.

Higher (more positive) responses $H_y(Q)$ indicate higher confidence that the true class label $L(Q)$ of pattern Q is y. To classify a query $Q \in \mathbb{X}$, we evaluate $H_y(Q)$ for all $y \in \mathbb{Y}$, and classify Q as belonging to the class y for which $H_y(Q)$ is maximized. More specifically, if we denote as $H(Q)$ the output of the multiclass classifier H for pattern Q, $H(Q)$ is defined as:

$$H(Q) = \text{argmax}_{y \in \mathbb{Y}} H_y(Q) \ . \tag{4.2}$$

At runtime, given a pattern Q to classify, the standard approach is to apply all OVA classifiers H_y, and identify the y such that H_y gives the strongest response. Clearly, this approach has complexity linear to the number of classes. Our goal in this paper is to show that the strongest-responding classifier H_y can be found efficiently, using vector search methods, without needing to evaluate $H_y(Q)$ for all y. This topic is addressed in the next sections.

4.3 Reduction to Nearest Neighbor Search

The core observation underlying our method is that, for JointBoost-based multi-class recognition, both test patterns and OVA classifiers can be represented as vectors, specifying points on the surface of a hypersphere. Finding for a test pattern Q the strongest-responding OVA classifier H_y can be done by doing nearest neighbor search on those points.

In particular, we will map both OVA classifiers and test patterns into a $(d+2)$-dimensional vector space, where d is the number of weak classifiers that are used to define the OVA classifiers. We denote by $V(Q)$ and $V(H_y)$ respectively the vectors corresponding to test pattern Q and OVA classifier H_y. In defining this mapping, we will explicitly ensure that all resulting vectors have the same norm. Ensuring that all $V(H_y)$ and $V(Q)$ have the same norm will be used in reducing the problem of finding the winning OVA classifier for each Q to the problem of finding the nearest neighbor of $V(Q)$ among all $V(H_y)$.

We begin by defining the vector $V(H_y)$ corresponding to each OVA classifier H_y:

$$V(H_y) = (\alpha_{y,1}, \ldots, \alpha_{y,d}, k_y, c_y) \ . \tag{4.3}$$

In the above equation, $\alpha_{y,m}$ and k_y are the weights and class-bias terms used in Equation 4.1, and c_y is a class-specific quantity that ensures that all $V(H_y)$ have the same Euclidean norm.

Quantity c_y can be determined as follows: first, we need to identify what the maximum norm of any $V(H_y)$ would be if we set all c_y to zero:

$$N_{\max} = \sqrt{\max_{y \in \mathbb{Y}}[(\sum_{m=1}^{d} \alpha_{y,m}^2) + k_y^2]} \ . \tag{4.4}$$

Then, we define c_y as:

$$c_y = \sqrt{N_{\max}^2 - [(\sum_{m=1}^{d} \alpha_{y,m}^2) + k_y^2]} \ . \tag{4.5}$$

By defining c_y this way, it can easily be verified that the Euclidean norm of every $V(H_y)$ is equal to $N_{\max}$.

Now we can define the vectors corresponding to test patterns. In particular, given a pattern $Q \in \mathbb{X}$, we define an auxiliary vector $V_{\text{orig}}(Q)$, and the vector of interest $V(Q)$, as follows:

$$\begin{aligned} V_{\text{orig}}(Q) &= (h_1(Q), \ldots, h_d(Q), 1, 0) \ , & (4.6) \\ V(Q) &= \frac{N_{\max} V_{orig}(Q)}{\|V_{orig}(Q)\|} \ , & (4.7) \\ & & (4.8) \end{aligned}$$

where $\|V\|$ denotes the Euclidean norm of V, and h_m are the weak classifiers used in Equation 4.1.

Using these definitions, Equation 4.2 can be rewritten as follows:

$$\begin{aligned} H(Q) &= \text{argmax}_{y \in \mathbb{Y}} H_y(Q) & (4.9) \\ &= \text{argmax}_{y \in \mathbb{Y}} (V_{orig}(Q) \cdot V(H_y)) & (4.10) \\ &= \text{argmax}_{y \in \mathbb{Y}} (V(Q) \cdot V(H_y)) \ , & (4.11) \end{aligned}$$

where $V_1 \cdot V_2$ denotes the dot product between vectors V_1 and V_2. To justify the above lines, we first observe that the $(d+2)$-th coordinate of $V(H_y)$, which is set to c_y, does not influence $V_{\text{orig}}(Q) \cdot V(H_y)$, since the $(d+2)$-th coordinate of each $V_{\text{orig}}(Q)$ is set to zero. Therefore, it can be easily verified that, for all H_y, $H_y(Q) = V_{\text{orig}}(Q) \cdot V(H_y)$. Also, since $V(Q)$ is just a scaled version of $V_{\text{orig}}(Q)$, the same H_y that maximizes $V_{orig}(Q) \cdot V(H_y)$ also maximizes $V(Q) \cdot V(H_y)$.

We will now take one additional step, to show that maximizing the dot product between $V(H_y)$ and $V(Q)$ is the same as minimizing the Euclidean distance between $V(H_y)$ and $V(Q)$. That can be easily shown, by using the fact that both $V(Q)$ and $V(H_y)$ are vectors of norm $N_{\max}$, because the dot product and the Euclidean distance for vectors of norm $N_{\max}$ are related as follows:

$$\|V(Q) - V(H_y)\|^2 = 2N_{\max}^2 - 2(V(Q) \cdot V(H_y)) \ . \tag{4.12}$$

The above equation can be easily derived as follows:

$$\|V(Q) - V(H_y)\|^2 = \tag{4.13}$$

$$= (V(Q) - V(H_y)) \cdot (V(Q) - V(H_y)) \tag{4.14}$$

$$= (V(Q) \cdot V(Q)) + (V(H_y) \cdot V(H_y)) - \tag{4.15}$$

$$2(V(Q) \cdot V(H_y)) \tag{4.16}$$

$$= 2N_{\max}^2 - 2(V(Q) \cdot V(H_y)) \ , \tag{4.17}$$

using the fact that $(V(Q) \cdot V(Q)) = (V(H_y) \cdot V(H_y)) = N_{\max}^2$.

By combining this result with that of Equation 4.11, it follows readily that:

$$H(Q) = \operatorname{argmin}_{y \in \mathbb{Y}}(\|V(Q) - V(H_y)\|) \ . \tag{4.18}$$

This result means that, given a test pattern Q, finding the strongest-responding OVA classifier H_y is reduced to finding the nearest neighbor of $V(Q)$ among all vectors $V(H_y)$. The next section describes how to use that fact for speeding up multiclass recognition.

4.4 A Simple Vector Indexing Scheme

So far we have established that, in order to classify via JointBoost a test pattern Q, it suffices to find the nearest neighbor of $V(Q)$ among all vectors $V(H_y)$. Clearly, vectors $V(H_y)$ can be computed off-line and stored in a database. The importance of reducing JointBoost-based classification to nearest neighbor search is that a vast array of vector indexing methods can be used to speed up this search, such as, e.g., the methods in [57, 58, 41, 59, 60].

In order to illustrate the computational savings that can be obtained by treating JointBoost-based classification as a nearest neighbor search problem, we have implemented a simple and easy-to-use vector indexing method that is based on principal component analysis (PCA) [70]. Since the set of vectors $V(H_y)$ is computed off-line, we can use those vectors for an additional off-line step, where PCA is used to identify the principal components of those vectors and the corresponding projection matrix Φ. Given a test pattern Q, its vector $V(Q)$ can be projected to $\Phi(V(Q))$ online, and then $\Phi(V(Q))$ can be compared to the projections $\Phi(V(H_y))$ of the vectors corresponding to classifiers H_y.

PCA can easily be used within a filter-and-refine retrieval framework [43]: given a user-defined integer parameter p, filter-and-refine works as follows:

- Input: A test pattern Q, and its vector representation $V(Q)$.
- Filter step: Compute the projection $\Phi(V(Q))$ to the lower-dimensional space, and find the nearest neighbors of $\Phi(V(Q))$ among the set of all $\Phi(V(H_y))$. Keep the top p nearest neighbors, where p is a user-defined parameter, as mentioned above.
- Refine step: For each of the top p nearest neighbors, compute $H_y(Q)$.
- Output: Return the H_y yielding the strongest response $H_y(Q)$, among the H_y's evaluated during the refine step.

As long as $d' \ll d$ (where d' is the number of dimensions of $\Phi(V(Q))$, and d is the number of weak classifiers), the filter step is significantly faster than simply applying all H_y to Q. At the refine step we do evaluate some classifiers H_y, but, if $p \ll d$, these classifiers are only a small subset of the entire set of OVA classifiers.

4.4.1 Guarantees of Accuracy

We should note that the simple filter-and-refine method outlined above does not guarantee achieving the same accuracy as brute-force search. In other words, it does not guarantee that the H_y retrieved at the refine step will be truly the one that we would have identified if we had simply evaluated $H_y(Q)$ for all y. However, our filter-and-refine method can be easily modified to guarantee achieving the same accuracy as brute-force search.

More specifically, we can utilize the fact that PCA is a *contractive* mapping, meaning that the Euclidean distance between $\Phi(V(Q))$ and $\Phi(V(H_y))$ is guaranteed to be not greater than the Euclidean distance between $V(Q)$ and $V(H_y)$. When the filter step estimates distances based on a contractive mapping, it is well-known that the refine step can be defined in a way that guarantees finding the true nearest neighbor. Details on that topic can be found at [43]. In our experiments we found that, although we use a filter-and-refine version that does not guarantee finding the nearest neighbor 100% of the time, the accuracy that we obtained in practice was so high that it was not worth implementing a more complicated version.

We should note that a large variety of vector and metric indexing methods also guarantee finding the correct nearest neighbor, e.g., the methods in [59, 60, 42]. Such methods can easily be integrated into the filter step of our method.

4.5 Classification Time Complexity

Given a test pattern Q, the time that it takes to classify Q using the proposed method can be decomposed to the following costs:

- Weak classifier cost: The cost of computing $h_m(Q)$ for each weak classifier h_m. This takes time $O(d)$, where d is the number of weak classifiers. For JointBoost, it is empirically observed in [56] that d tends to increase logarithmically with the number of classes, so this time cost should become a negligible fraction of total time as the number of classes increases.
- Projection cost: The cost of computing the PCA projection $\Phi(V(Q))$. If Φ projects from $d+2$ dimensions to d' dimensions, this takes time $O(d^2)$, and becomes a negligible fraction of total time as the number of classes increases, assuming that, as mentioned earlier, d scales logarithmically with the number of classes.
- Filter cost: The cost of measuring Euclidean distances between $\Phi(V(Q))$ and $\Phi(V(H_y))$ for each H_y. This takes time $O(d'|\mathbb{Y}|)$, where $|\mathbb{Y}|$ is the number of classes. This is still linear to the number of classes, but we can obtain a big constant factor of savings if $d' \ll d$. We should also note that several methods exist for sublinear nearest neighbor search in vector spaces, including the popular LSH method [58], and such methods can be easily integrated into our method to achieve time sublinear to the number of classes.
- Refine cost: The cost of evaluating $H_y(Q)$ for each H_y selected at the filter step. This takes time $O(dp)$, where p is the number of classifiers H_y selected at the filter step. As shown in our experiments, typically $p \ll |\mathbb{Y}|$, so the refine cost is much smaller than simply evaluating $H_y(Q)$ for each H_y.

4.6 Experiments

The datasets used in our experiments were generated from two original datasets: a dataset of hand images, where the task is to estimate the handshape and the 3D orientation, and the Face Recognition Grand Challenge (FRGC) Version 2 dataset [71] of 2D face images. Using these datasets, we compare the proposed method to brute-force search, which is the standard way of classifying patterns using OVA classifiers, not only for JointBoost, but in general for methods based on boosting and support vector machines [55, 64]. We also compare our method with ClassMap [63], a method that can be used to speed up OVA-based classification. We only used ClassMap embeddings trained using AdaBoost, as specified in [63], because these embeddings were shown in [63] to outperform other versions of ClassMap embeddings.

4.6.1 Datasets

4.6.1.1 The Hand Dataset

This dataset contains hand images of 81 basic hand shapes defined in American Sign Language (ASL). There are 30 different out-of-plane view angles for each shape, and 20 in-plane rotations for each out-of-plane view, for a total of $81 \times 30 \times 20 =$ 48,600 hand pose classes. The training examples used for each class were 150 synthetic images, generated using Poser 5 [72].

For each synthetic hand image, cluttered background from random real images was added to the regions outside the hand silhouette. From each hand image, a histogram-of-oriented-gradient (HOG) feature vector [73] of dimension 2,025 was extracted. The image was normalized to 48 by 48 pixels, which was divided into cells of size 6 by 6, with neighboring cells overlapping by half. For each cell, nine edge orientation bins were evenly spaced between 0 to 180 degrees. Bins in each cell were normalized with the surrounding 3 by 3 cells. All the bins from all the cells were vectorized into a feature vector of 2025 feature components for a hand sample. Each weak classifier h_m is a feature stump, completely specified by parameters f_m and t_m, that checks whether the f_m-th HOG feature is greater than t_m or not. JointBoost selected 3,000 weak classifiers after training on this dataset.

For evaluation, we used a synthetic test set, disjoint from the training set, and consisting of 281 synthetic hand images (chosen randomly among images from all 48,600 classes). In addition to the synthetic hand images, we also used a second test set of 992 real hand images, collected from 7 subjects and with cluttered background. Because of the difficulties in visually estimating the 3D hand orientation on an image, we assigned to each hand image three different class labels (out of the 48,600 possible class labels). Each of those three class labels corresponded to the same handshape and a 3D orientation within 30 degrees of the manually labeled orientation. The classification result is considered correct iff it is equal to one of those three labels.

4.6.1.2 The Face Dataset

This dataset contains all 2D face images in the FRGC-2 dataset [71], amounting to 36817 face images from 535 subjects (i.e., 535 classes). The original resolution of the face images was either 1704×2272, or 1200×1600. All images were converted to gray images and normalized to 100 by 100 pixels. A PCA space was learned from 4,000 uniformly sampled training faces of all the subjects. The features of face images were their projections on the top 2,509 PCA components, which accounts for 99.9% of the variance. JointBoost selected 10,000 weak classifiers after training on this dataset. Each weak classifier h_m is a feature stump, completely specified by parameters f_m and t_m, that checks whether the f_m-th PCA dimension is greater than t_m or not. For evaluation, we used 300 face images, that were chosen randomly, and excluded from the training set.

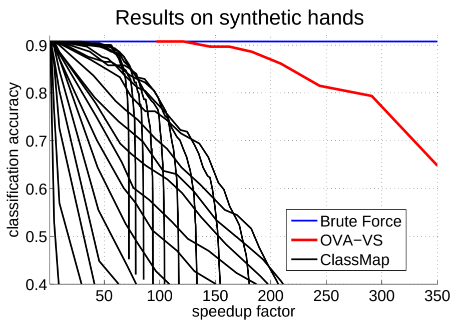


Figure 4.1. Accuracy vs. speed-up factor obtained by the proposed OVA-VS method, ClassMap, and brute force, on the test set of synthetic hand images. The brute force accuracy, which is a single value equal to 90.75%, is shown as a horizontal line..

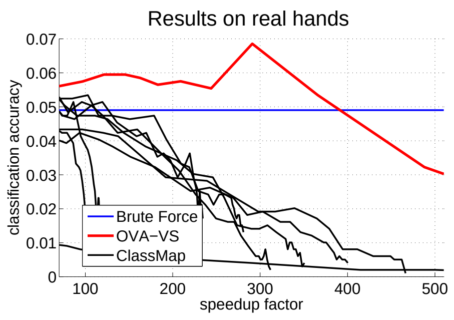


Figure 4.2. Accuracy vs. speed-up factor obtained by the proposed OVA-VS method, ClassMap, and brute force, on the test set of real hand images. The brute force accuracy, which is a single value equal to 4.9%, is shown as a horizontal line..

4.6.2 Results

Performance is measured in terms of speed-up factor with respect to brute force, and classification accuracy. The speed-up factor is the ratio between classification time using our method (or ClassMap) and classification time using brute-force search. By definition, brute force achieves a speed-up factor of 1. For the proposed method, the parameters that need to be chosen are d', i.e., the dimensionality of the lower-dimensional PCA space, and p, i.e., the number of OVA classifiers to be evaluated at the refine step. To reduce the number of free parameters to one, we decided to set for our method, in all experiments, $p = \frac{nd'}{d}$, where n is the number of classes. This

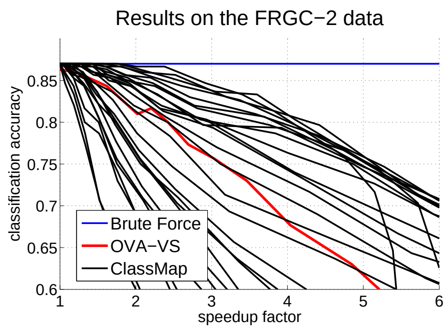


Figure 4.3. Accuracy vs. speed-up factor obtained by the proposed OVA-VS method, ClassMap, and brute force, on the test images of the FRGC-2 face dataset. The brute force accuracy, which is a single value equal to 87.0%, is shown as a horizontal line..

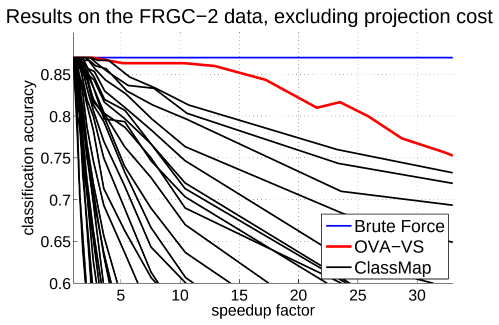


Figure 4.4. Accuracy vs. speed-up factor (ignoring the projection cost of OVA-VS and ClassMap) obtained by the proposed OVA-VS method, ClassMap, and brute force, on the test images of the FRGC-2 face dataset. The brute force accuracy, which is a single value equal to 87.0%, is shown as a horizontal line..

constraint is a simple choice that forces the filter and the refine step to have the same running time. The PCA projection matrix, for each dataset, was computed based solely on the vectors $V(H_y)$ of the OVA classifiers obtained for each dataset.

In presenting the results we refer to the proposed method as OVA-VS, an acronym for OVA-based classification using vector search.

4.6.2.1 Results on the Synthetic Hands Dataset

Figure 4.1 compares the performance of the proposed OVA-VS method, ClassMap, and brute-force search on the test set of 281 synthetic hand images. In Figure 4.1, we plotted a single performance curve for OVA-VS, obtained by constraining d' and p as specified above, and varying d'. In contrast, for ClassMap, we plotted a family of curves, each curve corresponding to a different embedding dimensionality, ranging from 1 to 90 dimensions, and to varying p. The single OVA-VS curve corresponds to much better accuracy vs. efficiency trade-offs than any of the results obtained for ClassMap. As a highlight, OVA-VS gave a speed-up factor of 120 over brute-force search for a classification accuracy of 90.75% (equal to that of brute-force search), whereas ClassMap gave a speed-up factor of only 29 for that accuracy. OVA-VS yielded this result for $d' = 12$ and $p = 194$.

Figure 4.2 shows results for the test set of 992 real hand images. Once again, the single OVA-VS curve corresponds to much better accuracy vs. efficiency trade-offs than any of the results obtained using different dimensionality and p parameters for ClassMap. OVA-VS gave a speed-up factor of 290 over brute-force search for a classification accuracy of 6.85%. The highest accuracy obtained for ClassMap was 5.3%, for a speedup factor of 105. We note that both ClassMap and OVA-VS attained accuracies higher than brute-force search for the real images. This is a somewhat curious result, that was also reported in the original ClassMap paper [63]. Intuitively, the lower-dimensional projection of OVA-VS and embedding of ClassMap can be seen as new features extracted from the output of the weak classifiers, and according to our results these new features lead to higher accuracy than the original weak classifiers.

Additionally, on the real hand images, OVA-VS yields an accuracy of 5.35%, still slightly better than that of brute force search, for a speedup factor of 365. ClassMap produces accuracies better than or equal to that of brute force search only for speedup factors less than or equal to 127.

It is worth noting that, even using brute force, classification accuracy drops significantly from 90.75% for the synthetic test images to 4.9% for the real hand images. The relatively low accuracy for the real hand images simply reflects the difficulty of estimating hand pose from a single image, due to the very large number of possible classes. Since there are 48,600 classes, a classification accuracy of 4.9% is still 2381 times higher than the accuracy of a random classifier. We should also note that even with this accuracy, the proposed hand pose estimation system can be useful for initializing a hand tracker based on particle filtering, where temporal integration can be used to significantly improve overall tracking accuracy.

4.6.2.2 Results on the FRGC-2 Dataset

Figure 4.3 plots the results attained with the proposed OVA-VS method, ClassMap, and brute force on the FRGC-2 dataset. We note that for an accuracy of 84.2%, which is 2.8% lower than that of brute-force search, OVA-VS achieved a speedup factor of only 1.6. As seen on the same figure, ClassMap achieved a speedup of 3.0 for the same accuracy of 84.2%, and a speedup of 1.8 for an accuracy of 87%, which is equal to the accuracy of brute-force search.

The main reason for the relatively small improvement in classification time attained by both OVA-VS and ClassMap is the relatively small number of classes in this dataset: only 535, compared to the 48,600 classes of the synthetic hands dataset. As a result, generating a single dimension of a PCA projection, or a single dimension of a ClassMap embedding, are operations that incur 1/535 of the cost of brute-force search, compared to 1/48600 for the hands dataset. In other words, the relatively small number of classes makes the projection cost defined in Section 4.5 more pronounced. As discussed in Section 4.5, the projection cost for our method becomes negligible as the number of classes becomes large, and this is also true for ClassMap.

Figure 4.4 plots classification accuracy vs. speedup factor, but ignores, in computing running times, the projection cost of generating PCA projections and ClassMap embeddings for the queries. This curve is representative of the performance we could expect if we had a much larger number of classes, that would make the projection costs negligible. In that case we see that the proposed OVA-VS method performs better than ClassMap, obtaining, for example, a speedup factor of 12.9 for a classification accuracy of 86.0%. For that same accuracy, ClassMap gives a speedup factor of 4.6.

The results that exclude the projection cost are promising, and indicate that we can expect significant classification time improvements from using our method in face recognition domains with tens of thousands of classes or even more. Unfortunately, we are not aware of any publicly available face dataset with such a large number of classes. To the best of our knowledge, no public dataset of 2D face images contains more classes than the FRGC-2 dataset, while still providing a sufficient number of training examples per class to learn OVA classifiers. While several important applications require face recognition in the presence of tens of thousands of classes or more (especially in homeland security and surveillance domains), security and privacy concerns make it difficult for such datasets to be made publicly available.

4.6.2.3 Summary of results

On both synthetic and real hand images, the proposed OVA-VS method significantly outperformed ClassMap and led to speedups of two orders of magnitude compared to brute force with no losses in classification accuracy. On the face dataset, the performance of OVA-VS was hampered by the projection cost, which was relatively high due to the relatively small number (535) of classes. When we excluded the projection cost from the overall running time, to get a picture of the expected performance when the number of classes reaches 10000 or more, OVA-VS again significantly outperformed ClassMap, and gave speedups of over one order of magnitude compared to brute force, with very little reduction (from 87% to 86%) in classification accuracy.

4.7 Discussion

We have shown that multiclass recognition using JointBoost can be reduced, at runtime, to a nearest neighbor search problem in a vector space. This reduction allows the use of a wide array of vector indexing methods for speeding up multiclass recognition. In our experiments, we have shown that a very simple indexing method, that uses PCA to select a few candidate nearest neighbors, works very well in practice and achieves, for the hands dataset, speedups of two orders of magnitude with no loss in classification accuracy, compared to brute-force search. Our method outperforms ClassMap in the hands dataset, and if we ignore the projection cost (which would be negligible if we had a significantly larger number of classes), our method also outperforms ClassMap on the faces dataset.

In comparing our method with ClassMap, it is worth noting that ClassMap defines a *lossy* vector representation of OVA classifiers H_y and patterns Q. Therefore, if V_{CM} is the vector mapping defined by ClassMap, the nearest neighbor of $V_{\text{CM}}(Q)$ among all $V_{\text{CM}}(H_y)$ is not guaranteed to correspond to the H_y maximizing $H_y(Q)$. In contrast, the method proposed in this paper defines a *lossless* vector representation, where the nearest neighbor always corresponds to the strongest-responding classifier.

We should emphasize that, instead of PCA, any other vector indexing method can also be integrated in the filter step of the proposed method. Several vector indexing methods guarantee finding the correct nearest neighbor for each query. Using such methods for the filter step of our algorithm guarantees that classification accuracy using our method will always equal that of brute-force search.

Naturally, an interesting topic for future exploration is to try a larger number of vector indexing methods, in order to identify methods that tend to work well in practice within the proposed framework. Also, as the proposed method is only applicable to JointBoost-based classification, it will be interesting to investigate whether similar methods can also be designed for other types of large margin classifiers, such as support vector machines. Progress in this area can lead to a broader theory of how to integrate database indexing methods with general large margin methods, so as to achieve scalable classification time complexity in domains with a very large number of classes.

CHAPTER 5

DIMENSIONALITY REDUCTION FOR EFFICIENT SEARCH IN PROTEIN SEQUENCES

This chapter focuses on the problem of efficiently searching a large database of strings in order to find the ones most similar to a given query string, under the edit distance. More specifically, we are interested in the special case where the query string is long (hundreds of letters or longer). An application that can benefit from progress on this topic is similarity search in biological databases (where each string is the encoding of a protein or a DNA sequence). Another potential application is searching databases of literature or code, to identify possible cases of plagiarism. However, in our discussion we will limit ourselves to the problem of searching protein databases.

In particular we are interested in finding, for a specific query string Q and range δ, all the database strings whose distance from Q is less than δ. For convenience, we typically express δ as a percentage of the query length Q. For example, if the query length is 1000 and $\delta = 15\%$, we want to retrieve all database strings whose distance from Q is less than or equal to 150.

Evaluating the similarity between two strings under the edit distance is computationally expensive, as it takes time linear to the product of the lengths of the two strings. A common framework for speeding up similarity searches is the filter-and-refine approach. For each query, the filter step has a fast, but less accurate, way of identifying a small set of candidate objects from the database. The refine step performs expensive similarity evaluations to determine which of the candidates match the desired search.

The distribution of pairwise distances between long protein strings has low variance, meaning that such distances are very likely to have values relatively close to a certain mean distance. This property, which is a manifestation of the well-known curse of dimensionality, makes tree-like index structures ([76], [77]), which prune the search space using the triangle inequality, ineffective for this problem. Alternative methods can use an index structure (e.g., q-gram inverted lists [74]) or a simpler similarity measure (e.g., alphabet reduction [75]).

Our main contribution is defining an informative embedding of the strings into a new space, in which distances can be measured orders of magnitude faster. The embedding is based on appearances of a few substrings among the protein strings. In particular we select a set of codewords, and for each protein string we generate a new string that shows the occurrences of these codewords in the original string. Such a representation captures the similarity information since similar strings are more likely to have common substrings. The distance between the embedded objects is highly informative regarding the distance between the original strings. At the same time, the embedding maps each string into a much shorter string (assuming some appropriate choices in defining the embedding), so that computing distances in the embedded space is much faster.

5.1 Related Work

One method for speeding up string comparisons is alphabet reduction ([75]). Essentially groups of letters are collapsed to one symbol (for example all the odd letters of the alphabet are replaced by a 1 and all the even letters are replaced by a 0). Promising results are reported in [75], using that method in conjunction with reference-based embeddings. However, alphabet reduction does not reduce the length of the strings, and thus it takes similar time to compute the distance between two strings, regardless of whether we have reduced the alphabet or not. In contrast, the method proposed here drastically reduces the length of the strings, thus dramatically speeding up the computation of approximate distances.

Q-gram methods([74]) are based on the idea that if two sequences have a certain degree of similarity they must share a minimum number of subsequences (q-grams). In the off-line step the occurrence of each q-gram in each of the database objects is recorded in an inverted list. For a query search, all the q-grams of that query are extracted and their corresponding inverted lists are used to identify the database objects that share a minimum number of q-grams with the query. Next a refine step is performed to identify which of the candidates match the search requirements. This method is guaranteed to always return all of the objects that match the search criteria. One of the differences between this method and ours is that we do not use all of the possible substrings and that we preserve some of the location/ordering information from the original strings.

The actual implementation that we have used for Q-grams is from the publicly available Flamingo Package, [78]. That package uses a smart implementation of the edit distance that stops the building of the matrix if it can be easily determined that the distance will be larger than the required threshold. Such quick determinations can be obtained using letter counting, and also using partial distance estimates computed by building the lower and upper half of the distance matrix separately.

Another indexing method ([79]) uses a set of reference objects and the triangle inequality. The distances from the reference objects to the query and to the database objects are used in conjunction with the triangle inequality to determine if a database object is certainly within the required similarity range (that is it should be part of the search result), certainly outside of it (excluded from the search result) or a possible candidate. The resulting candidates are then evaluated using the edit distance and the search result set is updated. This method is also exact, rendering all the objects that match the search criteria. Its efficiency depends on the pruning power of the distances to the reference objects and is thus hindered by the low variance in the distribution of pairwise distances.

5.2 Method Description

Our method can be seen as a dimensionality reduction (DR) method. A specific implementation of our method is given in Figure 5.1. For each query we identify, in a greedy way, a set $\mathbb{E}$ of codewords, consisting of the the top t most frequent substrings of length 2. The set of codewords has the property that there is no pair of strings that have overlapping suffix-prefix (that is, for any two strings $R, S \in \mathbb{E}$, no prefix of R is a suffix of S, and no prefix of S is a suffix of R). Next we embed the query and the database according to the occurrence of these codewords, as follows. Each codeword is assigned a corresponding new letter, a sequence is parsed from left to right, and as soon as one of the codewords is found it is replaced by its corresponding letter. All the other parts of the original sequence are deleted. Thus, the resulting string is significantly shorter than the original string. For example, the embedding of string $babfcde$ according to the codeword set $\mathbb{E} = \{ab, cd\}$ (where ab is mapped to 1 and cd is mapped to 2) is 12.

In the target space of the embedding, we perform brute-force search for embedded database strings that are within a certain range of the embedding of the query string. The range δ' that we use in the target space depends on δ, i.e. the range that the user has specified in the original space, and is equal to δf, where f is a scaling factor greater than one. For example, if $\delta = 15\%$ and $f = 2$, then $\delta' = 30\%$. δ' is higher than δ to account for the cases where the distance in the embedded space is a higher percentage of the embedded query. This can easily happen because the embedded query and database strings are drastically shorter than the original strings. While we expect similar strings to map to similar embedded strings, the loss of information incurred by the embedding can lead to higher distances as percentages of query length. Good values for f can be estimated in a straight-forward manner, using a training set of queries. This training set can be chosen as a subset from the database, or alternatively we can use randomly generated strings.

The candidates identified using the filter step are then refined in the original space.

```
Input:
  Query sequence $Q \in \mathbb{X}$.
  Database set $\mathbb{S} \subset \mathbb{X}$.
  Search range $\delta$.
  Number of codewords, $t$ .
  Adjustment scale factor for the target space $f$.
Output:
  Result set, $\mathbb{N} \subset \mathbb{S}$, of database objects within the search range.

1. Find the top $t$ most frequent codewords of $Q$, $\mathbb{E}$.
2. Embed $Q$ according to $\mathbb{E}$.
3. $d = \lfloor |Q| * \delta \rfloor$
4. For each $X \in \mathbb{S}$:
    (Length Filter) If abs$(X - Q) > d$:
      reject $X$.
    Else:
      Embed $X$ according to $\mathbb{E}$: $x = E(X)$
      If $\mathcal{D}(x, q) \leq \delta \cdot |q| \cdot f$, then $X$ is a candidate. Refine the search:
          If $\mathcal{D}(X, Q) \leq \delta \cdot |Q|$, add $X$ to the result set, $\mathbb{N}$.
```

Figure 5.1. Dimensionality Reduction Search Algorithm.

5.2.1 Theoretical Analysis

Definition 1. *A set of codewords $\mathbb{E}$ is a set of strings such that, for any strings $R, S \in \mathbb{E}$, no prefix of R is a suffix of S, and no prefix of S is a suffix of R.*

Theorem 1. (Contractiveness) *Let X, Y be two strings, and $\mathbb{E}$ be a set of codewords as defined above. Let x and y be the $\mathbb{E}$-embeddings of X and Y. Then $\mathcal{D}(X,Y) \geq \mathcal{D}(x,y)$.*

Proof: Let $T(X,Y)$ be an optimal matching between X and Y. We represent X and Y as a sequence of matching and non-matching subsequences. The non-matching subsequences can be empty. In particular $X = N_1P_1N_2P_2 \ldots N_mP_mN_{m+1}$ and $Y = N'_1P'_1N'_2P'_2 \ldots N'_mP'_mN'_{m+1}$. The P subsequences represent the parts of the strings that are perfectly one-to-one matched: $P_i = P'_i \neq \emptyset$ for all $1 \leq i \leq m$. The N subsequences represent the remaining pieces of the strings. For any N_i, N'_i pair, at most one of the substrings can be empty. They are matched using insertion, deletion and substitution. In this optimal matching, the cost of matching the P sequences is zero. Therefore the cost of the match is: $D(X,Y) = \sum_{i=1}^{m+1} max\{|N_i|, |N'_i|\}$.

Now we distinguish two cases:

1. The codewords do not cross the boundaries between N and P sequences. In this case let $x = n_1p_1 \ldots p_nn_{m+1}, y = n'_1p'_1 \ldots p'_nn'_{m+1}$ be a representation that remembers the original matching. That is n_i are the pairs extracted from N_i and so on for all the other subsequences (P, P', N'). Let $t(x,y)$ be a matching, in the embedded space, that matches n_i with n'_i and p_i with p'_i for all $1 \leq i \leq m+1$. The cost of matching the (p, p') pairs is zero since they are identical (they were created from identical pairs (P, P')). Since the embedding is based on non-overlapping pairs, for all i, $|n_i| \leq |N_i|/2$. The cost to match (n, n') pairs is

$$cost(t(x,y)) = \sum_{i=1}^{m+1} max\{|n_i|, |n'_i|\} = \sum_{i=1}^{m+1} max\{\frac{|N_i|}{2}, \frac{|N'_i|}{2}\} =$$

$$\frac{1}{2}\sum_{i=1}^{m+1} max\{|N_i|,|N_i'|\} \leq \frac{D(X,Y)}{2}.$$

Since $t(x,y)$ is not necessarily the optimal matching we have:

$$D(x,y) \leq cost(t(x,y)) \leq \frac{D(X,Y)}{2}.$$

2. The codewords do cross some boundaries between N and P. This case can be eliminated by not allowing codewords to have overlapping suffixes or overlapping prefixes. In the embedded string, assign the pairs that cross boundaries to the non-matching segments. Each pair is replaced by a letter and there was at least one letter in the non-matching sequence of the original string that was mismatched. Thus the cost of matching the non-matching pieces in the embedded string is at most equal to that of matching them in the original string:

$$D(x,y) \leq cost(t(x,y)) \leq D(X,Y).$$

□

Proposition 5. *The conditions for a legal set of codewords $\mathbb{E}$ from Definition 1 are necessary for proving that the embedding is contractive (in particular for satisfying the property that one change in the original string should generate at most one change in the embedded version of the string).*

Proof: Proof by contradiction: . Let $\mathbb{E} = \{ca, ac\}$ and $X = eaca$. The $\mathbb{E}$-embedding of X is $x = 2$ (only ac is found). Let $Y = caca$. The $\mathbb{E}$-embedding of Y is $y = 11$. The lower bound property is not satisfied: $\mathcal{D}(X,Y) = 1 \ngeq 2 = \mathcal{D}(x,y)$. □

5.3 Experiments

5.3.1 Datasets

For our experiments we use data from the UniProt dataset ([80]) of protein sequences. It has 530264 strings over an alphabet of 25 letters. The protein strings have variable length ranging from 2 to 35213. For our experiments, we have used a dataset that uses a total of 28155 sequences (100 for test, 100 for validation and 27955 for the database). The set of 28155 sequences we have used consists of all sequences in the UniProt dataset that have lengths between 801 and 1600. All 3 sets (test, validation, database) are disjoint, and membership in each dataset was randomly assigned.

5.3.2 Methods

In the experiments we compare our method with other methods that have been proposed in the literature for this problem. In particular, the methods that we evaluate are the following.

- Dimensionality reduction (DR): our method, as described in this chapter.
- Reference-based embedding: the method described in [79], that uses distances to reference objects, and the triangle inequality, to quickly identify a small set of candidate matches. For this method we have built a new index for each search range.
- Q-Grams: the method described in [74], that uses inverted indexes of q-gram occurrences to quickly identify candidate matches. We use Flamingo Package code, [78], that the authors of [74] have made publicly available.

We compare our method with Q-grams and the reference-based embeddings. For each method we report that cost as a percentage of the brute force cost and the runtime. Our method is not exact and thus we also report the recall percentage there.

5.3.3 Evaluation Measures

- Retrieval accuracy: Here we measure, out of all the database objects that are within the desired range from the query, the percentage that are successfully retrieved by the system. When we report cumulative results on a set of queries, we sum up the total number of correct results that the system retrieves and we divide it by the total number of correct results that should be retrieved. Reference-based embeddings and Q-grams are exact methods, and they guarantee an accuracy of 100%. Our method does not guarantee 100% accuracy and thus it is important to document the accuracy that our method actually achieves at each experiment.
- Runtime: The actual average runtime per query that it takes for each method to produce results, measured over our set of 100 test queries. Runtimes were measured on a 2GHz Intel Xeon (QuadCore, but our experiments used a single core) with 4GB of RAM, under Windows 7.
- Retrieval cost estimate: One limitation of measuring efficiency using running times is that those times can depend significantly on particular aspects of the hardware, such as memory, cache size, bus speed and so on. Running times also depend on the efficiency of the implementation, compiler optimizations and choice of programming language. As an alternative platform-independent measure, we use a more theoretical estimate, where we try to use worse-case estimates for our method and best-case estimates for the competitors. We

believe that these numbers help obtain a clearer picture of the efficiency that our method achieves compared to the competitors.

Our measure of retrieval efficiency is reported as a percentage of brute-force search. Brute-force has to compute entries on dynamic programming tables. If $|Q|$ denotes the length of the query and $|\mathbb{X}|$ denotes the sum of lengths of database strings, then the number of entries that must be computed in these dynamic programming tables is $|Q| * |\mathbb{X}|$. Since all methods we evaluate have a refine step, we measure, at each experiment, a quantity that we denote as x', which is the sum of lengths of all database strings that are considered at the refine step. Fraction $x'/|\mathbb{X}|$ is a lower bound of the computational cost of a method, as a percentage of brute-force. It is a lower bound because it does not take into account the cost of any processing outside the refine step (such as the cost of the filter step).

For Q-grams and reference-based embeddings we report $x'/|\mathbb{X}|$ as our estimate of retrieval efficiency. As noted above, this is a favorable method for those methods, as it ignores all cost outside of the refine step.

For our method we add to $x'/|\mathbb{X}|$ two additional quantities: an estimate of the embedding cost and an estimate of the filter cost. For the embedding cost we consider every letter of the embedded database strings to cost as much as computing an entry in a dynamic programming table. For the filter cost, the computation is more straightforward, as the filter step measures edit distances in the embedded space. We simply count the total number of entries in the dynamic programming tables computed during the filter step. So, overall, the efficiency of our method is measured in units of dynamic programming table entries, and we simply divide that cost by the cost of brute-force.

To report the retrieval efficiency over a set of queries, we simply report the average of the retrieval efficiencies, $x'/|\mathbb{X}|$, attained for the individual queries.

5.3.4 Implementation Choices

5.3.4.1 Dimension Reduction Method

To implement our method we need to make certain choices. Here we document the choices we have made and the process for making those choices.

- Length of codewords: Here we have only considered codewords of length two. As we obtained, in our opinion, satisfactory results, we did not consider longer codewords.
- Number of codewords: We have experimented with using two, three, and four codewords. Most of the times four codewords worked better in our validation set, so that is the setting we have used in our results on the test set, unless otherwise specified.
- Scaling factor, f: As a reminder, the scaling factor f is used in equation $\delta' = f\delta$, where δ is the user-specified search range, and δ' is the search range that our system uses in the embedded space. Larger values of f, bring accuracy closer to 100%, but also bring the search cost closer to the cost of brute force search. To choose f for each experiment, we used our validation set, and we identified the smallest value of f that produced over 99% retrieval accuracy. The reason we chose 99% was that we wanted close to 100% accuracy, but at the same time we did not want the value of f to be determined by a few query outliers. Obviously, while the chosen value of f produces 99% accuracy on the validation set, we still need to measure the actual retrieval accuracy obtained by those

values on the test set. As the results show, the accuracy obtained on the test set does not differ much from the accuracy obtained on the validation set.

5.3.4.2 Q-Grams Method

For Q-grams, we made the following choices in our experiments:

- We used the DivideSkip merging algorithm because it was reported as being the most efficient one in [74] and it also performed better in a few initial experiments with our datasets.
- We used a filter tree with a length filter and fan of 10 for 4-grams. We ran experiments for $\delta = .2$ (since this was the first more challenging search range for this method) with q-gram lengths between 2 and 7. We have found 4 to be the optimal value. We have also experimented with different fanout values, but they did not seem to make a difference.

5.3.4.3 Reference-Based Embedding

We have used 3000 random sample objects from the database for references, but we have excluded the cost of computing the embedding and filtering the results in the reported cost of this method. Both the theoretical cost and the runtime presented in this chapter include only the refine step.

5.3.5 Results

The brute force runtime in the implementation of our method was 298.72 seconds per query. Our method was implemented in Java. The brute force runtime for Q-grams (with all the optimizations mentioned in section 5.1) was 248.13 seconds per query.

Table 5.1 shows the retrieval cost estimate, as defined in section 5.3.3, for all methods, for searches within ranges $5\%, 10\%, 15\%, 20\%, 20\%$ and 30% of the query length. We note that our method achieves significantly lower costs than the competitors for $\delta = 25\%$ and $\delta = 30\%$. For values of δ between 5% and 20%, q-grams are actually more efficient. We also note that, while our method does not guarantee 100% retrieval accuracy (which the competitors do guarantee), the actual accuracy obtained does not fall below 99% and is actually measured at 100% for most of the cases.

Table 5.2 shows the runtimes for all methods, again for searches within ranges $5\%, 10\%, 15\%, 20\%, 20\%$ and 30% of the query length. The results here closely resemble the retrieval cost results, except that our method almost ties q-grams for $\delta = 20\%$. Q-grams produce the best results for δ values of 5%, 10%, 15%, and our method produces the best results for $\delta = 25\%$ and $\delta = 30\%$.

Table 5.3 shows, for our method, for each search range, the cost and runtime as reported in the previous tables, and also the scale factor f and resulting δ' search range in the embedded space.

Table 5.4 shows details for the reference-based embedding method. In particular, for different search ranges, we show the retrieval cost, the percentage of database objects that were pruned using the embedding, and the runtime (with parameters $m = k = 2000$ chosen randomly as discussed in section 5.3.4) The embedding cost reflects computing the distances between the query and the 2000 reference objects and it is 7.15% of the brute force cost.

Finally, Table 5.5 shows the difference in length between the original query and database strings and their embedded versions. We note that, on average, embedded queries were about 20 times shorter than the original queries, and that the embedded database strings were about 50 times shorter than the original database strings.

Table 5.1. Retrieval cost estimate (as defined in 5.3.3) for all methods. Parameter δ indicates the distance range (expressed as a fraction of query length) within which we want to retrieve database matches.

δ	Retrieval accuracy	Dimensionality reduction	Reference-based embedding	Q-grams
5%	100%	0.1272%	0.4848%	0.0076%
10%	100%	0.1513%	3.3781%	0.0144%
15%	100%	0.1770%	6.1162%	0.0213%
20%	100%	0.2285%	9.4995%	0.1207%
25%	99.14%	0.4179%	13.1420%	77.9949%
30%	100%	6.9830%	17.4084%	80.6616%

Table 5.2. Runtimes (as defined in 5.3.3) for all methods. Parameter δ indicates the distance range (expressed as a fraction of query length) within which we want to retrieve database matches. All runtimes are reported in seconds.

δ	Retrieval accuracy	Dimensionality reduction	Reference-based embedding	Q-grams
5%	100%	0.14	1.52	0.05
10%	100%	0.24	10.35	0.09
15%	100%	0.35	18.82	0.14
20%	100%	0.48	29.32	0.47
25%	99.14%	0.82	40.47	139.37
30%	100%	10.95	53.59	190.71

The fact that the embedded queries are not shortened by as large a factor as the database strings is expected; the embedding used for every query (and that is applied to the database, so as to process that query) is query-specific, and identifies the most frequently occurring codewords in the query.

5.4 Discussion and Future Work

The experimental results demonstrate that, for higher values of δ, our method produces significantly lower costs and runtimes than the competitors. One price that we pay with our method, compared to the competitors, is the loss of guarantee of

Table 5.3. Experimental results for dimensionality reduction. The number of codewords is 4. δ' is the search range used in the embedded space: $\delta' = \delta \cdot scale$.

δ (%)	Recall (Accuracy)	Cost (%BF)	Runtime (seconds)	Scale	δ' (%)
5%	100%	0.1272%	0.14	3.6	18.00%
10%	100%	0.1513%	0.24	3.5	35.00%
15%	100%	0.1770%	0.35	2.7	40.50%
20%	100%	0.2285%	0.480	2.2	44.00%
25%	99.14%	0.4179%	0.82	1.9	47.50%
30%	100%	6.9830%	10.950	1.9	57.00%

Table 5.4. Experimental results for reference-based embeddings competitor method. The embedding cost is 7.15% of the brute-force cost, and is not included in the reported cost. The reference objects were chosen randomly from the database objects. No optimization was performed ($m = k = 2000$). When a query is presented, each database object is evaluated based on the $k = 2000$ triangle inequalities.

δ	Cost	Pruned	Runtime
5%	0.49%	99.51%	1.52
10%	3.38%	96.62%	10.35
15%	6.12%	93.88%	18.82
20%	9.50%	90.5%	29.32
25%	13.14%	86.86%	40.47
30%	17.41%	82.59%	53.59

Table 5.5. The average length of test and database sequences in the original space and in the embedded spaces produced by dimensionality reduction using the top 4 codewords (DR-4). The % columns show the ratio between the new size and the original size (e.g. in DR-4% first row, 5.08% = 51.09/1036.07 * 100).

data	original	DR-4	DR-4%
test	1036.07	51.09	5.08%
database	1035.97	21.02	2.03%

100% retrieval accuracy. At the same time, we believe that this price can be an acceptable trade-off in several domains, given the significant runtime savings that our method achieves.

It will be interesting to explore directions for improving the performance of our method. One approach may be to implement multiple filter steps, in place of the single filter step that our method uses. These filter steps can be applied in sequence, so that each filter step is applied only on the candidates selected by the previous step, and each filter step does somewhat more work(e.g., by using more codewords) than the previous step, so as to prune away some more candidates.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

This thesis described methods for similarity search in multimedia databases. We have illustrated the different types of uses that such search can have, with applications such as sign language recognition, time series analysis, face recognition, and search in biological databases.

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 two methods, for two different domains that improve retrieval accuracy, by producing results that are more in line with human expectations. For sign language recognition, we have proposed a similarity measure that produces good results for the target application of looking up the meaning of individual signs. For time series analysis, we have proposed the MSM metric, which in several public datasets produces better nearest neighbor classification accuracy than existing alternatives.

With respect to improving retrieval efficiency, the thesis has proposed two methods that significantly improve efficiency in two target domains. One method, focuses on the topic of speeding up recognition of a large number of classes. A main contribution there has been to actually show that under certain conditions, multiclass recognition, becomes mathematically equivalent to similarity-based search. This equivalence allows the use of off-the-shelf existing similarity indexing methods to speed-up multiclass recognition.

Finally, for the problem of searching biological databases of strings, we have proposed a dimensionality reduction method, that significantly speeds up similarity searches under the edit distance for strings with length of several hundred characters or more. Our method is based on mapping such strings to much shorter representations, based on occurrences of a small number of codewords. The proposed method is simple to implement, outperforms existing competitors, and gives particularly good results for edit distance ranges between 10% and 25% of the query length, where existing methods typically break down.

While the proposed four methods have extended the state of the art in their respective target domains, there are still important challenges remaining in the broad area of similarity search of multimedia databases.In many cases, such as sign language recognition, our state-of-the-art similarity measures still fall significantly short of human accuracy. Furthermore, the complexity of searching very large databases, remains prohibitive for several interesting applications, such as searching the World Wide Web for images and video of interest. We hope that the contributions made in this thesis move the state of the art somewhat closer, to addressing the important challenges that lie ahead. We are also interested in exploring, in future work, some of these remaining challenges.

REFERENCES

[1] A. Stefan, H. Wang, and V. Athitsos, “Towards automated large vocabulary gesture search,” in *Conference on Pervasive Technologies Related to Assistive Environments (PETRA)*, 2009.

[2] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, Q. Yuan, and A. Thangali, “The American Sign Language lexicon video dataset,” in *IEEE Workshop on Computer Vision and Pattern Recognition for Human Communicative Behavior Analysis (CVPR4HB)*, 2008.

[3] E. Keogh, “The UCR time series data mining archive. http://www.cs.ucr.edu/ eamonn/tsdma/index.html,” 2006. [Online]. Available: http://www.cs.ucr.edu/~eamonn/TSDMA/index.html

[4] H. Wang, A. Stefan, S. Moradi, V. Athitsos, C. Neidle, and F. Kamangar, “A system for large vocabulary sign search,” in *Workshop on Sign, Gesture and Activity (SGA)*, 2010.

[5] A. Stefan, V. Athitsos, and G. Das, “The Move-Split-Merge metric for time series,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, accepted in March 2012, to appear.

[6] A. Stefan, V. Athitsos, Q. Yuan, and S. Sclaroff, “Reducing JointBoost-based multiclass classification to proximity search,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[7] R. A. Tennant and M. G. Brown, *The American Sign Language Handshape Dictionary.* Washington, DC: Gallaudet U. Press, 1998.

[8] H. Lane, R. J. Hoffmeister, and B. Bahan, *A Journey into the Deaf-World*. San Diego, CA: DawnSign Press, 1996.

[9] J. Schein, *At home among strangers*. Washington, DC: Gallaudet U. Press, 1989.

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

[11] B. Bauer, H. Hienz, and K.-F. Kraiss, "Video-based continuous sign language recognition using statistical methods." in *International Conference on Pattern Recognition*, 2000, pp. 2463–2466.

[12] P. Dreuw, T. Deselaers, D. Keysers, and H. Ney, "Modeling image variability in appearance-based gesture recognition," in *ECCV Workshop on Statistical Methods in Multi-Image and Video Processing*, 2006, pp. 7–18.

[13] T. Starner and A. Pentland, "Real-time American Sign Language recognition using desk and wearable computer based video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1371–1375, 1998.

[14] C. Vogler and D. N. Metaxas, "Parallel Hidden Markov Models for American Sign Language recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 1999, pp. 116–122.

[15] Y. Cui and J. Weng, "Appearance-based hand sign recognition from intensity image sequences." *Computer Vision and Image Understanding*, vol. 78, no. 2, pp. 157–176, 2000.

[16] Y. Ke, R. Sukthankar, and M. Hebert, "Efficient visual event detection using volumetric features," in *IEEE International Conference on Computer Vision (ICCV)*, vol. 1, 2005, pp. 166–173.

[17] S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell, "Hidden conditional random fields for gesture recognition." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2006, pp. 1521–1527.

[18] T. Kadir, R. Bowden, E. Ong, and A. Zisserman, "Minimal training, large lexicon, unconstrained sign language recognition," in *British Machine Vision Conference (BMVC)*, vol. 2, 2004, pp. 939–948.

[19] J. Zieren and K.-F. Kraiss, "Robust person-independent visual sign language recognition." in *Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, vol. 1, 2005, pp. 520–528.

[20] C. Valli, Ed., *The Gallaudet Dictionary of American Sign Language.* Washington, DC: Gallaudet U. Press, 2006.

[21] H. Cooper and R. Bowden, "Learning signs from subtitles: A weakly supervised approach to sign language recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 2568–2574.

[22] A. Farhadi, D. A. Forsyth, and R. White, "Transfer learning in sign language," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[23] A. Bobick and J. Davis, "The recognition of human movement using temporal templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 23, no. 3, pp. 257–267, 2001.

[24] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2247–2253, 2007.

[25] G. Yao, H. Yao, X. Liu, and F. Jiang, "Real time large vocabulary continuous sign language recognition based on OP/Viterbi algorithm," in *International Conference on Pattern Recognition*, vol. 3, 2006, pp. 312–315.

[26] M. Jones and J. Rehg, “Statistical color models with application to skin detection,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999, pp. I:274–280.

[27] H. Rowley, S. Baluja, and T. Kanade, “Rotation invariant neural network-based face detection,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1998, pp. 38–44.

[28] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, “Fast subsequence matching in time-series databases.” in *ACM International Conference on Management of Data (SIGMOD)*, 1994, pp. 419–429.

[29] K.-P. Chan and A. W.-C. Fu, “Efficient time series matching by wavelets,” in *IEEE International Conference on Data Engineearing (ICDE)*, 1999, pp. 126–133.

[30] Y. Moon, K. Whang, and W. Han, “General match: a subsequence matching method in time-series databases based on generalized windows.” in *ACM International Conference on Management of Data (SIGMOD)*, 2002, pp. 382–393.

[31] Y. Moon, K. Whang, and W. Loh, “Duality-based subsequence matching in time-series databases.” in *IEEE International Conference on Data Engineering (ICDE)*, 2001, pp. 263–272.

[32] T. Argyros and C. Ermopoulos, “Efficient subsequence matching in time series databases under time and amplitude transformations.” in *International Conference on Data Mining*, 2003, pp. 481–484.

[33] D. Rafiei and A. O. Mendelzon, “Similarity-based queries for time series data.” in *ACM International Conference on Management of Data (SIGMOD)*, 1997, pp. 13–25.

[34] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. R. Kaeli, "Subsequence matching on structured time series data." in *ACM International Conference on Management of Data (SIGMOD)*, 2005, pp. 682–693.

[35] E. Keogh, "Exact indexing of dynamic time warping," in *International Conference on Very Large Databases (VLDB)*, 2002, pp. 406–417.

[36] M. Vlachos, D. Gunopulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *IEEE International Conference on Data Engineering (ICDE)*, 2002, pp. 673–684.

[37] L. Latecki, V. Megalooikonomou, Q. Wang, R. Lakämper, C. Ratanamahatana, and E. Keogh, "Elastic partial matching of time series," in *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, 2005, pp. 577–584.

[38] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *ACM International Conference on Management of Data (SIGMOD)*, 2005, pp. 491–502.

[39] L. Chen and R. T. Ng, "On the marriage of lp-norms and edit distance," in *International Conference on Very Large Databases (VLDB)*, 2004, pp. 792–803.

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

[41] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, pp. 517–580, 2003.

[42] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 311–321.

[43] G. Hjaltason and H. Samet, "Properties of embedding methods for similarity searching in metric spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 25, no. 5, pp. 530–549, 2003.

[44] M. Morse and J. Patel, "An efficient and accurate method for evaluating time series similarity," in *ACM International Conference on Management of Data (SIGMOD)*, 2007, pp. 569–580.

[45] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "FTW: fast similarity search under the time warping distance," in *Principles of Database Systems (PODS)*, 2005, pp. 326–337.

[46] N. Brisaboa, O. Pedreira, D. Seco, R. Solar, and R. Uribe, "Clustering-based similarity search in metric spaces with sparse spatial centers," in *SOFSEM 2008: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science, V. Geffert, J. Karhumaki, A. Bertoni, B. Preneel, P. Navrat, and M. Bielikova, Eds. Springer Berlin / Heidelberg, 2008, vol. 4910, pp. 186–197.

[47] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French, "Clustering large datasets in arbitrary metric spaces," in *IEEE International Conference on Data Engineering (ICDE)*, 1999, pp. 502–511.

[48] P. Indyk, "A sublinear time approximation scheme for clustering in metric spaces," in *Proceedings of Annual Symposium on Foundations of Computer Science (FOCS)*, 1999, pp. 154–159.

[49] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[50] C. M. Bishop, M. Svensén, and C. K. I. Williams, "GTM: The generative topographic mapping," *Neural Computation*, vol. 10, no. 1, pp. 215–234, 1998.

[51] C. Faloutsos and K. I. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *ACM International Conference on Management of Data (SIGMOD)*, 1995, pp. 163–174.

[52] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–374, 2000.

[53] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[54] V. Vapnik, *The nature of statistical learning theory.* Springer-Verlag New York, Inc., 1995.

[55] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: a unifying approach for margin classifiers," *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2000.

[56] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 29, no. 5, pp. 854–869, 2007.

[57] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

[58] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *International Conference on Very Large Databases (VLDB)*, 1999, pp. 518–529.

[59] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "The A-tree: An index structure for high-dimensional spaces using relative approximation," in *International Conference on Very Large Databases (VLDB)*, 2000, pp. 516–526.

[60] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *International Conference on Very Large Databases (VLDB)*, 1998, pp. 194–205.

[61] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, vol. 2, pp. 263–286, 1995.

[62] J. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAGS for multiclass classification," in *NIPS*, 2000, pp. 547–553.

[63] V. Athitsos, A. Stefan, Q. Yuan, and S. Sclaroff, "ClassMap: Efficient multiclass recognition via embeddings," in *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[64] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *Journal of Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[65] E. J. Ong and R. Bowden, "A boosted classifier tree for hand shape detection," in *Face and Gesture Recognition*, 2004, pp. 889–894.

[66] D. Gavrila and V. Philomin, "Real-time object detection for "smart" vehicles," in *IEEE International Conference on Computer Vision*, 2001, pp. 87–93.

[67] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla, "Hand pose estimation using hierarchical detection." in *ECCV Workshop on Human Computer Interaction*, 2004, pp. 105–116.

[68] A. Agarwal and B. Triggs, "Recovering 3D human pose from monocular images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 1, pp. 44–58, 2006.

[69] T. E. de Campos and D. W. Murray, "Regression-based hand pose estimation from multiple cameras," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 2006, pp. 782–789.

[70] I. Jolliffe, *Principal Component Analysis.* Springer-Verlag, 1986.

[71] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 947–954.

[72] *Poser 5 Reference Manual*, Curious Labs, Santa Cruz, CA, August 2002.

[73] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 886–893.

[74] C. Li, J. Lu, and Y. Lu, "Efficient merging and filtering algorithms for approximate string searches," *International Conference on data Engineering (ICDE)*, 2008.

[75] P. Papapetrou, V. Athitsos, G. Kollios, and D. Gunopulos, "Reference-based alignment in large sequence databases," *Proceedings of the Very Large Database Endowment (PVLDB)*, vol. 2, no. 1, pp. 205–216, 2009.

[76] C. Traina, A. J. M. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," *International Conference on Extending Database Technology (EDBT)*, pp. 51–65, 2000.

[77] M. R. Vieira, C. Traina, F. J. T. Chino, and A. J. M. Traina, "Dbm-tree: A dynamic metric access method sensitive to local density data," *Brazilian Symposium on Databases (SBBD)*, pp. 163–177, 2004.

[78] A. Behm, R. Vernica, S. Alsubaiee, S. Ji, J. Lu, L. Jin, Y. Lu, and C. Li, "UCI Flamingo Package 4.0," http://flamingo.ics.uci.edu/releases/4.0/, 2010.

[79] J. Venkateswaran, D. Lachwani, T. Kahveci, and C. Jermaine, "Reference-based indexing of sequence databases," in *International Conference on Very Large Databases (VLDB)*, 2006, pp. 906–917.

[80] http://www.ebi.ac.uk/uniprot/.

BIOGRAPHICAL STATEMENT

Alexandra Stefan was born in Bucharest, Romania, in 1979. She received her B.S. degree in Computer Science and Mathematics from the University of Bucharest, in 2002. She received her M.A. in Computer Science from Boston University in 2008, and her Ph.D. degree in Computer Science from the University of Texas at Arlington in 2012. Her research areas are computer vision, pattern recognition, and data mining, with focus on applications in gesture and sign language recognition.