

# A Weighted Distance Measure for Calculating the Similarity of Sparsely Distributed Trajectories

Pekka Siirtola, Perttu Laurinen, Juha Röning  
Intelligent Systems Group, Department of Electrical and Information Engineering  
University of Oulu  
Finland  
email: {pesiirto, perttu, jjr}@ee.oulu.fi

## Abstract

*This article presents a method for the calculating similarity of two trajectories. The method is especially designed for a situation where the points of the trajectories are distributed sparsely and at non-equidistant intervals. The proposed method is based on giving different weights to different points: points that are close to each other get smaller weights than the points that do not have neighbors nearby. The effectiveness of the method was tested with 12 data sets generated from two benchmark data sets. The classifying accuracy of the proposed similarity measure was compared with three other methods, such as dynamic time warping, and it was noted that the new proposed method classifies instances mainly more accurately and faster than the other three methods.*

## 1 Introduction and related work

Measuring the similarity or distance of trajectories is one of the key problems in the area of time series data mining and it is also widely studied. It has been applied into many different areas of research, such as medicine [4], industry [2] and economics [11].

Dynamic time warping (DTW) is found to be an efficient method for calculating the similarity of trajectories of different lengths [1]. It can also be used in situations where two trajectories vary in time. DTW ignores both global and local shifts in the time dimension. An improved version of the algorithm is introduced in [3]. The article combines DTW and uniform scaling. Uniform scaling is a technique that allows global scaling of a time series. By using a combination of these techniques it is possible to find the most similar subsequence, under the dynamic time warping distance, of any time series. A faster version of DTW (Fast-DTW) is presented in [14]. FastDTW gives an approxima-

tion of the dynamic time warping distance and it has linear time and space complexity.

Several Markov model-based similarity measures are presented in [10]. These methods can be used to measure the distance of two trajectories of different length. The similarities of two time series are not calculated from the original sequences, instead the original time series are transformed into vector sequences in the phase space, and state-transition sequences are also constructed. These sequences are used to calculate the distance of the trajectories. Methods were tested using a Cylinder-Bell-Funnel data set and the results show that these methods work better than Euclidean distance.

A non-metric method for measuring similarity based on least common subsequences (LCSS) is presented in [15]. The method gives more weight to those parts of the sequences that are similar, which produces good results with noisy data. In fact the method is very resistant to noise and gives better results than dynamic time warping when tested with fuzzy data. LCSS can be used only in two- and three-dimensional spaces. It is possible to calculate the similarity of trajectories of different lengths using LCSS.

Many similarity measuring applications are based on compressing data using SAX [7] and then calculating the distances. SAX is the first symbolic presentation of a time series and in [9] it was used for the first time to find similar patterns in a time series. The similarity measuring algorithm used in [9] is based on defining the distance between different symbols and then calculating similarity.

The similarity measure presented in this article is especially designed to calculate the similarity of trajectories of different length with points distributed non-equidistantly. Only matching of the whole time series and trajectories are considered in this article, because the subsequential matching can be studied using sliding window techniques [8].

**Table 1. Symbols and definitions**

Symbol	Definition
$traj_a$	Trajectory $a$
$traj_{a,i}$	The $i$ th point of trajectory $a$
$x(traj_{a,i})$	The $x$ -coordinate of $traj_{a,i}$
$n(traj_a)$	The number of points in trajectory $a$
$d(traj_a, traj_b)$	The distance between trajectories $a$ and $b$

## 2 Methods for calculating the similarity of trajectories with non-equidistant intervals

In this article it is assumed that the  $x$  axis is the time axis. Other symbols used in the article are presented in Table 1.

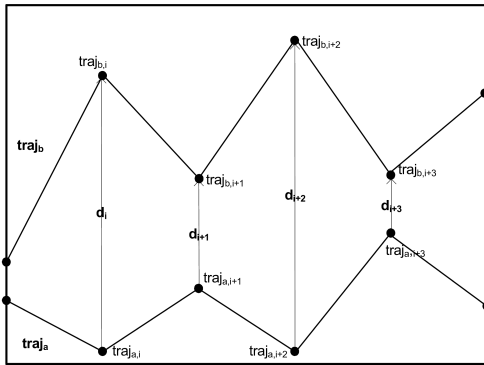
### 2.1 Problem statement and motivation

It does not sound like a difficult task to calculate the distance of two trajectories (or the similarity of two trajectories). In the case of Figure 1 the distance can be calculated for instance by simply calculating the Euclidean distance between points  $traj_{a,i}$  and  $traj_{b,i}$  for all  $i$  and considering the average distance of the points as the distance between the trajectories. In other words

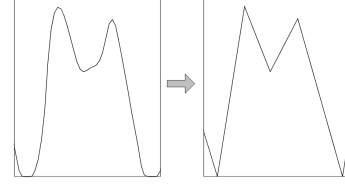
$$d(traj_a, traj_b) = \frac{1}{n} \cdot \sum_{i=1}^n d(traj_{a,i}, traj_{b,i}), \quad (1)$$

where  $n$  is the number of points of trajectories  $traj_a$  and  $traj_b$ .

Sometimes it is not feasible to use Equation (1), especially in cases where trajectories  $traj_a$  and  $traj_b$  contain a different number of points or the points are distributed at non-equidistant intervals. This kind of situation can happen if some points of the trajectories are lost for some reason or if the trajectories are simplified using compression.



**Figure 1. Calculating the similarity of trajectories of the same length.**



**Figure 2. Original data and compressed data.**

**Algorithm 1:** Algorithm for calculating the similarity between two trajectories presented in [6].

---

```

input : trajectories  $traj_a$  and  $traj_b$  of size  $n(traj_a)$  and  $n(traj_b)$ 
output : the distance between the trajectories,  $trajectorydistance$ 

set  $trajectorydistance$  to 0;
set  $smallestdistance$  to  $\infty$ ;

for  $i \leftarrow 1$  to  $n(traj_a)$  do
  for  $j \leftarrow 1$  to  $n(traj_b)$  do
    if  $d(traj_{a,i}, traj_{b,j}) < smallestdistance$  then
      | set  $smallestdistance$  to  $d(traj_{a,i}, traj_{b,j})$ ;
    end
  end
  increment  $trajectorydistance$  by  $smallestdistance$ ;
  set  $smallestdistance$  to  $\infty$ ;
end
set  $trajectorydistance$  to  $(trajectorydistance/n(traj_a))$ ;
return  $trajectorydistance$ ;

```

---

Figure 2 presents a compressing method that reduces the number of points of the trajectory in such a way that after compression, the trajectory contains only such points of the original time series where the value of the derivative is equal to zero. When trajectories are compressed using this method, they can be of different size after compressing even though they contain the same number of points as originally. Compression like this is sometimes necessary in order to speed up calculations. Calculations can be done faster from compressed trajectories because the less points, the faster the calculations.

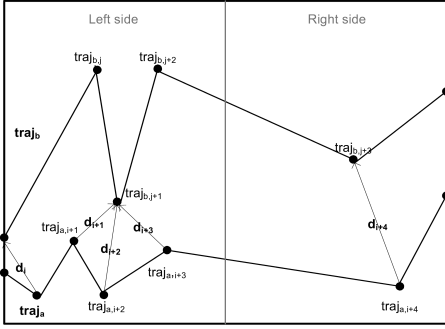
### 2.2 Intuitive method

Algorithm 1 presents an intuitive way to calculate the distance between trajectories with non-equidistant intervals.

When the distance of trajectories  $traj_a$  and  $traj_b$  is calculated (see Figure 3) using Algorithm 1, for each point  $traj_{a,i}$  of  $traj_a$ , the algorithm searches for a point in trajectory  $traj_b$  for which

$$d(traj_{a,i}, traj_{b,j}) < d(traj_{a,i}, traj_{b,k}),$$

where  $1 \leq k \leq n(traj_b)$ ,  $j \neq k$ . The average of these point-to-point distances is considered as the distance of the trajectories  $traj_a$  and  $traj_b$ .



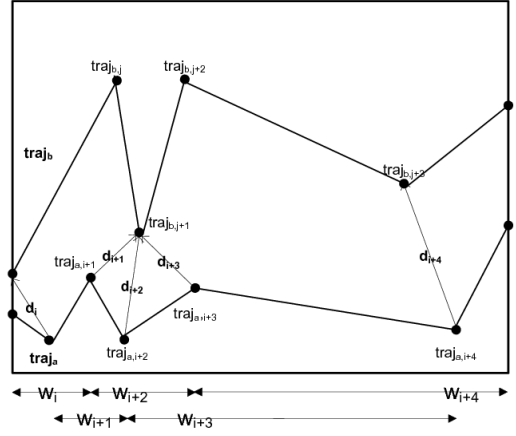
**Figure 3.** The left side includes more points than the right, so it has more weight on the similarity of these two trajectories.

### 2.3 Proposed method

The distance measure used in Algorithm 1 gives good results when the points of the trajectories are distributed at approximately equidistant intervals, but if the time series are sparse, a problem arises. The problem is the following: let us assume that the points of some trajectories are not located uniformly and in some areas of the time line there is a big group of points and there are also long intervals that do not include any points at all. Now when the similarity of these trajectories is calculated using Algorithm 1, an interval including several points has significantly more weight on the distance than interval that does not include points at all. For instance in the case of Figure 3, the left side of trajectory  $traj_a$  contains many more points than the right side, so the left side plays a more important role in the distance of  $traj_a$  and  $traj_b$  when Algorithm 1 is used.

One way to solve this problem is to do resampling. Resampling can be done by replacing the sparse trajectory with a trajectory that has points at equidistant intervals. To generate a trajectory that is not sparse, new points need to be added into places where they most likely would be located. The most probable places for the new points are on a line drawn between points that are closest to a new point. This method requires processing of data and for this reason it can be slow. A way to solve this problem without resampling is presented in Algorithm 2.

In Algorithm 2 the problem is solved by giving different weights to the different points: smaller weights to points that have neighbors nearby and bigger weight to points that are in areas where many points do not exist, Figure 4. This ensures that areas having several points does not have much greater impact on distance than other areas. Weights of the points cannot be directly proportional to the distance of neighbor points because much uncertainty is related to areas that do not contain points. For this reason points located at intervals that do not contain many points should get only a bit bigger weights than other areas. Weights  $w_i$  can



**Figure 4.** Algorithm 2 gives different weights to the different points: points that are close to each other get smaller weights than points that do not have neighbors nearby.

be calculated, for instance, by using the following equation:

$$w_i = \log(c_i + 1), \text{ where} \quad (2)$$

$$c_i = \begin{cases} \frac{x(traj_{a,2}) - x(traj_{a,1})}{denominator}, & i = 1, \\ \frac{x(traj_{a,i+1}) - x(traj_{a,i-1})}{2 \cdot denominator}, & 1 < i < n(traj_a), \\ \frac{x(traj_{a,n(traj_a)}) - x(traj_{a,n(traj_a)-1})}{denominator}, & i = n(traj_a), \end{cases} \quad (3)$$

where  $x(\cdot)$  is the  $x$ -coordinate of  $\cdot$  and

$$denominator = \frac{x(traj_{a,n(traj_a)}) - x(traj_{a,1})}{n(traj_a) - 1}. \quad (4)$$

Coefficients  $c_i$  tell how far away the neighbors of point  $traj_{a,i}$  are located compared with a case where the points of trajectory  $traj_a$  are located at equidistant intervals. In the case of  $1 < i < n(traj_a)$ , the variable  $denominator$  presented in Equation (3) is multiplied by two because in these cases point  $traj_{a,i}$  has two neighbors, one on both sides, but in the case of  $i = 1$  and  $i = n(traj_a)$  point  $traj_{a,i}$  has only one neighbor and the coefficient of the  $denominator$  is one. The variable  $denominator$  tells the average length of an interval of trajectory  $traj_a$ . In the Section 3, the weights for Algorithm 2 are calculated using Equation (2).

Note that if  $x(traj_{a,i+1}) - x(traj_{a,i}) = constant$ , then  $c_i = 1$  for all  $i$  and Algorithm 1 = Algorithm 2.

## 3 Experiments

Algorithm 2 was tested with artificial data (Cylinder-Bell-Funnel data set) and real-world data (Gun-Point data set), both data sets were provided by [5].

The results of Algorithm 2 were compared with the results given by Algorithm 1. Because Algorithms 2 and 1 are

---

**Algorithm 2:** Proposed algorithm for calculating the similarity between two trajectories.

---

**input** : trajectories  $tra_j_a$  and  $tra_j_b$  of size  $n(tra_j_a)$  and  $n(tra_j_b)$

**output** : the distance between the trajectories,  
 $trajectorydistance$

set  $trajectorydistance$  to 0;  
set  $smallestdistance$  to  $\infty$ ;

```

for  $i \leftarrow 1$  to  $n(tra_j_a)$  do
  set  $w_i$  to  $weight$ ;
  for  $j \leftarrow 1$  to  $n(tra_j_b)$  do
    if  $d(tra_{j_a,i}, tra_{j_b,j}) < smallestdistance$  then
      | set  $smallestdistance$  to  $d(tra_{j_a,i}, tra_{j_b,j})$ ;
    end
  end
  increment  $trajectorydistance$  by
   $weight \cdot smallestdistance$ ;
  set  $smallestdistance$  to  $\infty$ ;
end
set  $trajectorydistance$  to  $(trajectorydistance/n(tra_j_a))$ ;
return  $trajectorydistance$ ;

```

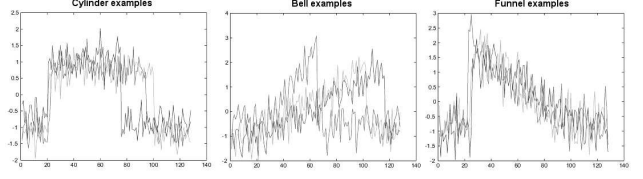
---

not symmetric, the distance of trajectories  $tra_j_a$  and  $tra_j_b$  was considered as

$$\max(d(tra_j_a, tra_j_b), d(tra_j_b, tra_j_a)) \quad (5)$$

which clearly is symmetric. This way Algorithms 2 and 1 satisfy three out of four requirements of metric space (non-negativity, identity of indiscernibles and symmetry). Only triangle inequality does not hold. The results from resampled data were also calculated. The similarities of the resampled trajectories were calculated using Algorithm 1. In addition, the results using DTW were also calculated. With every method used, point-to-point distances were calculated using Euclidean Distance.

To test the proposed algorithm in conditions where it was designed to work, Cylinder-Bell-Funnel (CBF) and Gun-Point (GP) data sets were modified so that they include intervals where points are missing. These data sets were created by randomly removing point sequences from the original data sets. Ten sequences were removed and the length of these sequences varied from six to ten points. Sequences were allowed to be located one on top of the other. So, if the length of the removed sequence was ten points, the number of removed points varied from ten to 100 points, ten in the case where all the sequences were one on top of other and 100 in the case where none of the sequences were one on the other. Data sets from which ten sequences of length six points were removed are called  $CBF_6$  and  $GP_6$ , sets from which ten sequences of length seven points were removed are called  $CBF_7$  and  $GP_7$  and so on. Distance measures were also tested with a compressed data set that contains only points where the derivative of the original data is zero. These data set are called  $CBF_{deriv}$  and  $GP_{deriv}$ .



**Figure 5.** Examples of a CBF data set.

The classification was done to each instance in turn. As a class label of instance  $tra_j_x$  of the test set we used the class label of its nearest neighbor from the training set.

### 3.1 Artificial data

CBF data set was originally proposed by [13] and it is used as benchmark data set in several articles to compare distance measures. The data set consists of three classes, cylinder, bell and funnel, see Figure 5. Each of these classes contains 310 instances, each instance contains 128 points, so the data set contains a total of 930 instances, which are generated from the following equations:

$$f(t) = (6 + \eta)X_{[a,b]}(t)(b - a)/(b - t) + \varepsilon(t)$$

$$b(t) = (6 + \eta)X_{[a,b]}(t)(t - a)/(b - a) + \varepsilon(t) \quad (6)$$

$$c(t) = (6 + \eta)X_{[a,b]} + \varepsilon(t),$$

$$X_{[a,b]} = \begin{cases} 1, & a \leq t \leq b, \\ 0, & \text{otherwise,} \end{cases}$$

where  $a \sim Unif(16, 32)$ ,  $(b - a) \sim Unif(32, 96)$  and  $\eta, \varepsilon \sim N(0, 1)$ .

The aim of the task was to classify instances into the correct classes with the help of a training set that included 10 instances from each class. This means the test set consisted of 900 instances, or 300 instances from each class. One hundred different training and test sets were generated from this data. Each of these 100 sets was generated by randomly choosing 10 instances from each class as a training set and leaving the rest of the data as a test set. This means that altogether 90,000 instances per CBF data set were classified.

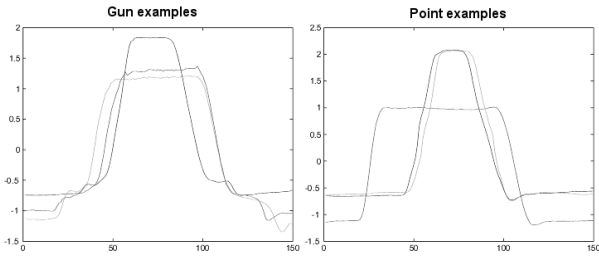
The results are in Table 2 and they show how different methods managed to classify data sets on average.

### 3.2 Real-world data

The Gun-Point data set was originally proposed by [12]. The data set consists of two classes, Gun-Draw and Point, see Figure 6. Each of these classes contains 100 instances and each instance contains 150 points. The data were created during one session using male and female actors. Actors performed two different actions: they draw and pointed a replicate gun (this action is called Gun-Draw) and their finger (this action is called Point) to predefined point. This

	Intuitive method	Proposed method	Resampling	DTW
$CBF_6$	97.88%	<b>98.30%</b>	96.50%	98.12%
$CBF_7$	98.24%	98.46%	97.39%	<b>98.56%</b>
$CBF_8$	93.13%	<b>94.64%</b>	91.42%	94.20%
$CBF_9$	90.91%	<b>92.21%</b>	87.34%	91.87%
$CBF_{10}$	88.08%	<b>89.74%</b>	85.01%	88.94%
$CBF_{deriv}$	99.02%	99.06%	98.62%	<b>99.47%</b>

**Table 2. Classification results for CBF data sets. The values show on average how the methods managed to classify data sets.**



**Figure 6. Examples of a Gun-Point data set.**

acting session was filmed and the data were extracted from video sequence by tracking actor’s right hand and transforming trajectory of it into a time series.

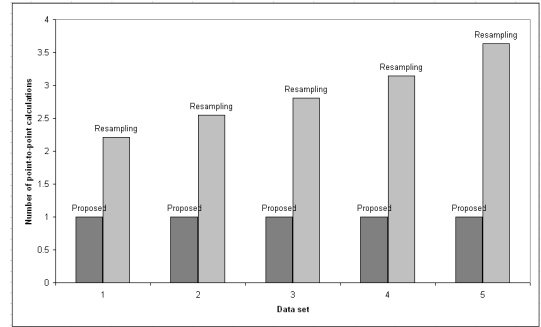
The aim of the task was to classify instances into the correct classes with the help of a training set that included 50 instances. The test set consisted of 150 instances, 75 instances from each class. One hundred different training and test sets were generated from this data. Each of these 100 sets was generated by randomly choosing 50 instances, 25 instances from each class, as a training set and leaving the rest of the data as a test set. Thus altogether 15,000 instances per Gun-Point-based data set were tried to classify.

The results are in Table 3 and they show how different methods managed to classify data sets on average.

The efficiency of proposed and resampling methods were compared, see Figure 7. Figure shows how many point-to-point distances are needed to calculate when resampling method is used instead of proposed method. The results of Figure 7 are calculated using faster version of Algorithm 1 presented in [6]. This faster algorithm reduces the point-to-point comparisons needed by Algorithm 1 to  $O(n)$  if the trajectories contain at least one increasing dimension. Same method was used to reduce the number of comparisons needed by Algorithm 2 to  $O(n)$ .

	Intuitive method	Proposed method	Resampling	DTW
$GP_6$	91.68%	92.37%	<b>93.48%</b>	91.12%
$GP_7$	89.56%	90.07%	<b>92.79%</b>	88.78%
$GP_8$	88.75%	89.69%	<b>91.71%</b>	88.49%
$GP_9$	85.55%	86.40%	<b>89.09%</b>	84.49%
$GP_{10}$	83.24%	85.47%	<b>87.77%</b>	82.81%
$GP_{deriv}$	93.97%	<b>95.63%</b>	84.72%	92.89%

**Table 3. Classification results for Gun-Point data sets. The values show on average how the methods managed to classify data sets.**



**Figure 7. Bars show how many point-to-point calculations resampling method requires compared to proposed method. 1 =  $GP_6$  data set. 2 =  $GP_7$  data set. 3 =  $GP_8$  data set. 4 =  $GP_9$  data set. 5 =  $GP_{10}$  data set.**

## 4 Discussion

The results presented in Section 3 shows that the results gained by using the proposed method are mainly better than the ones given by the three other, more commonly used, methods. Only resampling-based method seems to classify instances more accurately than proposed method in the case of real-world data. It was noted that the more points there are missing, the better the proposed method manages to classify instances compared with intuitive methods. This shows that the proposed method is worthwhile using instead of an intuitive method or DTW in the case of sparse data.

The proposed algorithm is better than the intuitive algorithm with every tested data set and better than resampling-based method with every tested data set generated from CBF data. It is noticeable how poorly the resampling method does compared with the other three tested methods when CBF data sets are classified. The reason for this can be that the resampling method makes too many assumptions about the distribution of the missing points. This kind of

method can work with data sets where there are no much variance between sequential points but in the case of the Cylinder-Funnel-Bell data set, there is much variation between two sequential points. The resampling method does not take this into account, and for this reason the results obtained with it are not good. When there is not much variation between sequential points, like in the case of Gun-Point data set, resampling method manages to classify instances better than any other similarity measure tested but it suffers from efficiency problems. The problem is that when resampling method is used, approximately three times more point-to-point distances are needed to calculate compared to proposed method, see Figure 7. Figure shows that the more points are missing, the faster proposed algorithm is compared to resampling method. Added to that resampling method requires lots of time to calculate point-to-point distances, time is also needed to preprocessing the data into resampled form before point-to-point distances can be calculated. Therefore resampling-based method is slow and not necessary can be used in all real-time applications or in applications which do not allow heavy computing.

The results show that when only small number of points is removed from the CBF data set, DTW and the proposed method can classify instances with almost similar accuracy. In fact DTW even manages to beat the proposed method when instances of  $CBF_7$  and  $CBF_{deriv}$  data sets are classified. But when more points are removed, proposed method starts to work better and it is more accurate than DTW. Note that DTW is better than intuitive algorithm with every tested data set. Still the proposed method, which is based on the intuitive method, classifies instances more accurately than DTW with four out of six tested CBF data sets. This shows how much the results improves when different weights are given to different points and the functioning of the proposed method. The results also show that in the case of real-world data the proposed method clearly outperforms DTW. In fact with Gun-Point-based data sets even intuitive method classifies instances more accurately than DTW.

The results achieved using the method presented in this article are not as good as the best classification results obtained with the data sets. This is a consequence of the fact that in this article classification was done using sparse data, while in other articles whole data sets were used. This of course dilutes the quality of the results and for this reason the results can not be directly compared.

Note that the introduced method is not algorithm-dependent and can be used together with other similarity algorithms, such as DTW, too.

## 5 Acknowledgment

The authors would like to thank the Finnish Funding Agency for Technology and Innovation and Infotech Oulu

for funding this work. P. S. would like to thank GETA (The Graduate School in Electronics, Telecommunications and Automation) for financial support and J. Sandhu for help.

## References

- [1] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pages 359–370, 1994.
- [2] J. Crossman, H. Guo, Y. Murphey, and J. Cardillo. Automotive signal fault diagnostics - part i: signal fault analysis, signal segmentation, feature extraction and quasi-optimal feature selection. *Vehicular Technology, IEEE Transactions on*, 52(4):1063–1075, 2003.
- [3] A. Fu, E. Keogh, L. Lau, and C. Ratanamahatana. Scaling and time warping in time series querying. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 649–660. VLDB Endowment, 2005.
- [4] S. Hirano and S. Tsumoto. Mining similar temporal patterns in long time-series data and its application to medicine. *ICDM*, 00:219, 2002.
- [5] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The ucr time series classification/clustering homepage .
- [6] P. Laurinen, P. Siirtola, and J. Röning. Efficient algorithm for calculating similarity between trajectories containing an increasing dimension. In *Artificial Intelligence and Applications*, pages 392–399, 2006.
- [7] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pages 2–11, 2003.
- [8] S. Park, W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 23–32, 2000.
- [9] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Proceedings of IEEE International Conference on Data Mining (ICDM'02)*, pages 370–377, Maebashi City, Japan.
- [10] Y. Qian, S. Jia, and W. Si. Markov model based time series similarity measuring. *Machine Learning and Cybernetics, 2003 International Conference on*, 1:278–283 Vol.1, 2003.
- [11] D. Rafiei and A. O. Mendelzon. Querying time series data based on similarity. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):675–693, 2000.
- [12] C. A. Ratanamahatana and E. Keogh. Making time-series classification more accurate using learned constraints. In *SDM*, 2004.
- [13] N. Saito and R. R. Coifman. Local discriminant bases. In A. F. Laine and M. A. Unser, editors, *Wavelet Applications in Signal and Image Processing II, Proc. SPIE 2303*, pages 2–14, 1994.
- [14] S. Salvador and P. Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *3rd Wkshp. on Mining Temporal and Sequential Data, ACM KDD '04*, Seattle, Washington.
- [15] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684, 2002.