ISSN:0975-9646

Tree Pruning Based Searching of Similar Trajectories

Gajanan S. Gawde[#1], Jyoti D. Pawar[#2]

[#1]*Computer Engineering Department, Goa College of Engineering*
Farmagudi – Ponda , Goa – India

[#2] *Department of Computer Science & Technology, Goa University*
Taleigao Plateau– Panaji , Goa - India

Abstract— **Edit distance measures such as DTW, EDR , ERP , LCSS are used to compare trajectories for similarity. These distance measures are defined recursivly. The tree structure of such distance measures have large number of nodes. The large number of nodes make such distance measures more costly in terms of time complexity. There is need to reduce the number of nodes of tree and makes it more faster. After analysing the tree structure of different resursive edit distance measures, we have found that there are many duplicate nodes present in the tree and these duplicate nodes can be eliminated. We have proposed two algorithms which eliminate the duplcate nodes and makes algorithm more efficient as compared to existing distance measures. The first algorithm, PruneMatrix prune the tree based on the duplicate nodes found in the tree and unique nodes are stored in the matrix. As an when duplicate node is found , it is inserted into the matrix. This technique has extra overhead of processing matrix as and when node is added or searched an thus reduces the performance. In order to eliminate this extra overhead , we have proposed second algorithm, PruneHash, identifies the duplicate nodes and store it in hash table instead of table. Hash table takes constant time to search node from the table, thereby increasing the performance. We have carried out extensive experimental study on different datasets and recorded the performance of existing and proposed techniques. Experimental results revealed that, PruneMatrix and PruneHash techniques are efficient compared to the existing methods, whereas PruneHash technique is faster compared to PruneMatrix.**

Keywords— **Edit Distances, Tre Pruning, Prune Matrix, Prune Hashing.**

I. INTRODUCTION

Large number of time series trajectories are generated by moving objects and such trajectories need to be processed to extract hidden information. Extracting hidden information from time series trajectories is very challenging task and also time consuming. Two time series trajectories are compared using edit distance measures for similarity. The similarity distance computed by edit distance will decide whether trajectories are similar or not. If the similarity distance is less or equal to some threshold value , then two trajectories said to be similar. Edit distance such as DTW,EDR,ERP and LCSS are basic and popular distance measures used to compare time series trajectories for similarity. These distance measures are defined recursivly as shown below:-

1. Dynamic Time Waring (DTW) measure:-

DTW (R, S) = dist (r1, s1) + min { (DTW (Rest (R), Rest (S)), DTW (Rest (R), S), DTW (R, Rest (S))) } (1)

2. Edit Distance with Real Sequence measure:-

EDR (R,S) = min { (EDR (Rest (R), Rest (S)) + Matchdist, EDR (Rest(R),S) + 1, EDR (R,Rest (S)) + 1) } (2)

3. Edit Distance with Real Penalty measure :-

ERP(R,S) = min{ (ERP(Rest(R), Rest(S)) + dist (r 1, s1), ERP(Rest(R),S) + dist (r1, g), EDR(R,Rest(S)) + dist (s1, g) } (3)

4. Longest Common Sub Sequence measure:-

LCSS(R,S) = max{(LCSS(Rest(R),Rest(S)) + 1, LCSS(Rest(R), S) + dist (r1, g),LCSS(R,Rest(S)) + dist(s1,g) } (4)

All the above distance measures discussed are recursive in nature. This recursive definition of distance measures is responsible for generating huge tree with large number of nodes. After careful study of tree generated , we have revealed that there are many duplicates nodes present in the tree. These nodes need to be eliminated in order to reduce the time complexity. Recursive function terminates when the length of either one of the two trajectories is zero and it returns the similarity value as the minimum of the arguments passed to the recursive function.

There are many applications of similarity search of time series trajectories.

1. Personal Security:- Every eminent personality is worried about his or her security. Each day they are under tension that someone is following or traking them. There is need to provide safey to such people by identifying possible threat from unknown people. Searching of similar trajectories will identify all the persons whose trajectories are almost similar to eminent person. If trajectories are matching, then for how many days such similarities are found and cross checked with threshold value. If number of count is greater than some threshold value, then warning message is send to eminent personality for possible threat and proper care can be taken to avoid possible attack in near future.

2. ECG of Patient:- Most of time , doctors are checking ECG of the patient manually and such practice is harmful since human being can make a mistake. Manually checking of ECG is need to be eliminated and there is need to make this process automatic. In automatic system, ECG of the patient can be compared with the normal ECG of the persons which is stored in the database. If the ECG of the patient is not matching with the normal ECG of the person , then there is some problem with the patient and he might be suffering from a heart problem. Our automatic system would compare time series trajectories of ECG and display final results to the doctor, and based on the output doctor would take appropriate action. If ECGs are not matching ,

then doctor might perform thorough investigation on the ECG to identify main cause of the problem.

3. Traffic Management:- Vehicles Trajectories can be used to manage the amount of traffic on the road. The flow of traffic can be analysed by capturing traffic passing on the road and in case of congestion , alternate solution can be proposed. RTO officers can used the traffic information and depute traffic police staff to control the flow of traffic.

4. Famous Routes:- Vehicles Trajectories can be used to identify popular routes in the given state or country. The trajectories of the vehicles can be compared with database trajectories to identify number of similar trajectories. If the number of the matched trajectories are above certain threhold value, then that route can be called as a popular route.

Contribution of our work in this paper is as follows:

1. We have proposed PruneMatrix Technique to prune the number of nodes of tree.
2. In order to reduce extra overhead of PruneMatrix, we have proposed PruneHash Technique to prune the number of nodes of the tree.
3. Experimental results revealed that, PruneMatrix and PruneHash are efficient compared to existing techniques, further, PruneHash is efficient compared to PruneMatrix technique.

The rest of paper is organised as follows. In section 2 we present related work. In section 3, we propose PruneMatrix and PruneHash algorithms to prune redundant nodes of tree. Section 4, provides the experimental results and finally, section 5 concludes the paper.

Symbols	Meaning
R	Time Series Trajectory
S	Time Series Trajectory
[M,N]	M is the length of R trajectory N is the length of S trajectory
PruneMatrix	Pruning tree using Matrix
PruneHash	Pruning tree using Hashing

TABLE 1: MEANING OF SYMBOLS USED

II. RELATED WORK

Searching of similar trajectories is challenging problem and there is constant demand for efficient distance measure. Quite a few researchs have been contributed toward this field by proposing distance measures. The pioneering work by [1] ,[2] to compare trajectories for similarity using Euclidean distance measure. Eucledian distance measure cannot be applied directly to compare trajectories and thus there is need of scaling and translation of trajectories. In [3] trajectories were scaled and translated efficiently and compared for similarity. Eucledian distance measure is sensitive towards noise and local time shifting. [4] introduced DTW to allow a time series trajectories to be stretched to provide better match with another trajectories. DTW distance measure was compared for the perfomance and DTW was enhanced using lower bounding distance measure with segmentation in [5]. [6] introduced bounded similarity query with the help of eucledian distance to identify similar time series. [7] have proposed Eros,Muse and Ropes distance measures to identify k-nearest neigbhour using principle component.

LCSS distance measure is used in [8] and [9] to compare trajectories for similarity. LCSS measure allows a variable length gap to be inserted during matching of trajectories and hence robust to the noise. Extended LCSS distance measure is used to compare trajectories from video in [10]. Initially trajectories were extracted from video and then subsequently compared for similarity using LCSS measure. Multi dimesional trajectories were compared for similarity using LCSS in [9]. In [11] LCSS distance measure was modelled with sigmoidal function to compares trajectories for similarity. Mostly index need to be created again and again and most of distance measures are modeled to support single distance measure. In [12], index multi dimensional trajectories supporting multiple distance measures such as LCSS , DTW are proposed and the index structure was design in a such a way that there was no need to rebuild index again and again. Various dimensionality reduction methods were investigated in [13] and proposed novel PAA technique to reduce dimensionality. In [14] figure out the extra computation done by LCSS and same is enhanced by fine tuning the threshold value.

Most of distance measures are time consuming and hence there is need to index such distance measures.Various indexing techniques [15-20] were proposed to improve the performance of distance measures. Lower bound technique is proposed to index the distance measure and compares trajectories for similarity in [19].[21] have enhanced the indexing method of DTW by modeling exact indexing. Human motions were efficiently indexed using bounding rectangle by [22]. In [23] authors have proposed technique o create index only once and multiple tranformations were applied instead of appliying indexing multiple times, thus improving the perfomance of distance measure.

[24] proposed landmark similarity distance measure to compares trajectories for similarity by considering landmark as human perception and comparing trajectories based on lanmarks. Distance measure is defined using geometric set with deterministic and random property in [25]. Trajectories were aggregated and compared for similarity in [26]. In [27], One way distance(OWD) measures were proposed to compare the similarity between trajectories. In [28], moving objects were detected using Gaussian Hermit moments(GHM) based on the trajectories and also it identify direction of objects. The gestures of human motion were recognised based on the trajectories generated by human body in [29]. Eigen tracker technique was proposed to detect the changing shapes of human gesture. Anomaly behaviour of crowded surveillance screnes were detected using statistical method in [30]. Popular routes were detected using similar trajectories matching in [31]. The popular routes were detected automatically without taking much inputs from the user and based on the probabilistic method.In [32], RICK framework was defined to identify popular routes from uncertain trajectories. In [33], the hot route was identified based on the traffic density on the road and flowscan framework was developed.

III. TREE PRUNING BASED TECHNIQUE

The recursive nature of edit distance measures are responsible for generating huge tree structure with large number of nodes. As the size of datasets is increasing, the number of nodes of tree is growing exponentially. For every call, three nodes are getting created and each node in turn makes a call to three more functions and which create three more nodes and this process continues till both input trajectories are processed. We have applied PruneMatrix and PruneHash to DTW distance measures and similarly can be applied to remaining distance measures.

The DTW distance measure algorithm is shown in algorithm 1. The DTW distance measure is defined as recursive function which call itself and for every recursive call it make three recursive calls. Each function call is consider as node of the tree. DTW recursive function generate tree structure with large number of nodes and each node of the tree is processed to compute similarity distance. DTW recursive function is discussed in detail in Algorithm 1.

Algorithm 1: DTW(I,J) Algorithm

Input: R and S : Input Trajectories
Output: Similarity Distance : dist
1) if I == N or J == M then
2) return 0
3) d= dist(I,J)
4) return
d+min{DTW(I++,J++),DTW(I,J++),DTW(I+,J)}

Consider the two input trajectories R={(1,1),(1,2)} and S={(3,1),(1,4)} of length 2 respectively. The tree of the recursive DTW distance measure for two input trajectories R and S is as shown in the Figure 1. Each node is represented with the index value of two trajectories such as i and j. When the call is made to the recursive DTW function, index value of i and j are updated. Initially the root node of the tree has initialise to to 0, indicating two trajectories are processed from first point. The nodes n2,n10 and n12 having the same value (1,1) and these nodes are duplicate nodes. There is no need to process such nodes three times. Such node can be processed only once and we can use its processed value for rest of the computation. Thus , we can prune the subtree of duplicate nodes without further processing.

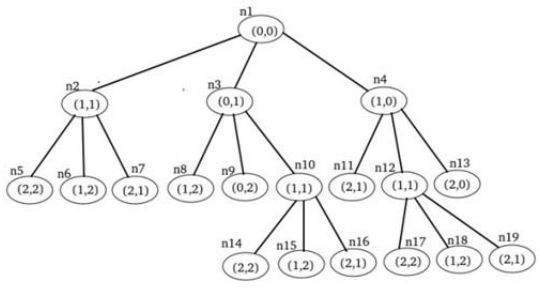

Figure 1 : Tree for Recursive Function

A. *PruneMatrix Algorithm*

The DTW recursive function makes three recursive calls and goes in the depth first search manner. Each recursive call is considered to be a node of the tree. So, every node is expanded as three child nodes of tree till leaf node is explored. After reaching leaf node , function backtrack to its previous explored node and try to explore remaining paths of the tree. Once root node is visited after backtracking all the nodes of the tree, the similarity distance is computed.

In PruneMatrix algorithm , every new node information is kept in the matrix and subsequently used to check if there are
any duplicate nodes down the tree. Starting with root node, root node is stored in the matrix and three more nodes are explored. The new node which is generated is first checked in the matrix ,if this node is not present, then it is added to the matrix and its child nodes are explored. If node is already present in the matrix , then this node is not added to matrix and it is marked as duplicate node. Since it is duplicate node , there is no need to explore its subtree and skip the entire subtree. Thus , we are reducing the number of nodes of tree.

Algorithm 2: PruneMatrix(I,J) Algorithm

Input: R and S : Input Trajectories
Output: Similarity Distance : dist
1) if I == N or J == M then
2) return 0
3) str1=str1 + R[I]
4) str2= str2 + S[j]
5) for each point k 2 count do
6) if Lookup[k].w1 ==str1 and Lookup[k].w2 ==str2 then
7) flag=1
8) distprune=Lookup[k].dist
9) if flag == 1 then
10) return distprune
11) d=dist(I,J)
12) td = d + min {PruneMatrix (I++,J++), PruneMatrix(I,J++), PruneMatrix(I++,J) }
13) Lookup[count++].w1=str1
14) Lookup[count++].w2=str2
15) Lookup[count++].dist=td
16) return td

The PruneMatrix algorithm is discussed in detail in Algorithm 2. Lines 1-2, define terminating condition for the PruneMatrix function. If length of either of the trajectory is equal to the length, then terminate the recursive call. Lines 5-10, search node in the matrix and if the node is found , then recursive call to PruneMatrix is not made. If the node is not found in the matrix , then new nodes are explored by making call to recursive PruneMatrix function. Line 12, makes call to recursive PruneMatrix function with three arguments. Lines 13-15, add new node to the matrix.

B. PruneHash Algorithm

PruneMatrix algorithm stores the information of every new node it encounter into the matrix. As the size of the matrix is grows large, we need to check if the node is present in the matrix. So, the time complexity of searching a node in matrix is of the order O(N) where N is the number of elements of the matrix. This is an extra overhead of the PruneMatrix algorithm and our algorithm slow down as the number of elements of time series trajectories increases. In order to remove this extra overhead , we have proposed hashing technique to reduce the time complexity of the searching. The time complexity of searching using hash function is of the order of constant i.e. O(C) where C is the constant.

We have proposed PruneHash Algorithm which eliminates the extra overhead faced by the PruneMatrix algorithm. In PruneHash algorithm , we have defined hash function using index i and j of two trajectories. Hash function is used to generate hash value to generate the index and this index is used to search the node in the matrix rather than performing sequential search. First, we are checking if new node is present in the Hash Table. If new node is not present, then we add this new node to the Hash Table with the help of index value computed using Hash function. The search operation on Hash Table is performed by using hash function in constant time. So , also to store the value in the Hash Table , the hash function is used. Using Hashing technique we are performing searching in constant time and therefore there is reduction of execution time at each node.

Algorithm 3: PruneHash(I,J) Algorithm

Input: R and S : Input Trajectories
Output: Similarity Distance : dist

```
1)  if I == N or J == M  then
2) return 0
3) str1=str1 + R[I]
4) str2= str2 + S[j]
5) str= str1 + str2
6) index = (I merge J) mod N
7)  if Lookup[index].w1 ==str1 and Lookup[index].w2
==str2  then
8) flag=1
9) distprune=Lookup[index].dist
10)  if flag == 1  then
11) return distprune
12) d=dist(I,J)
13) td = d + min{PruneHash(I++,J++),PruneHash(I,J+
+),PruneHash(I++,J) }
14) index = (I merge J) mod N
15) Lookup[index].w1=str1
16) Lookup[index].w2=str2
17) Lookup[index].dist=td
18) return td
```

The PruneHash algorithm is discussed in detail in Algorithm 3. Lines 1-2, define terminating condition for the PruneHash function. If length of either of the trajectory is equal to the length, then terminate the recursive call. Lines 5-11, search node in the hash table using hashing function and if the node is found , then recursive call to PruneHash is not made. If the node is not found in the hash table , then new nodes are explored by making call to recursive PrunePrune function. Line 13, makes call to recursive PruneHash function with three arguments. Lines 14-17, add new node to the hash table.

IV. EXPERIMENTAL STUDY

A. Characteristic of Trajectories Datasets

Character Trajectories:- This dataset is genetated by Lichman (2013), it consist of 2858 character samples, contained in the Three Dimensional Matrix. Each character sample is a 3-dimensional pen tip velocity trajectory. This is contained in matrix format, with 3 rows and T columns where T is the length of the character sample.

UJI Pen Character:- This dataset is generated by Lichman (2013), a character database by collecting samples from 11 writers. Each writer contributed with letters (lower and uppercase), digits, and other characters. Two samples have been collected for each pair writer/ character, so the total number of samples in this database version is 1364.

Optical Recognition Character:- This dataset is generated by UCI Lichman (2013), who have used preprocessing programs made available by NIST to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0 to 16. This reduces dimensionality and gives invariance to small distortions.

Real Time Characters :- This is our own synthetic dataset generated in real time with the help of 100 user. This dataset is generated using VB.net platform. Each user was asked to draw character on the display screen and same character was saved in the form of character trajectories. Each person had generated 50 number of different character trajectories.

B. System Configuration

Experimental study was carried out on Pentium V processor with 4GB of RAM and 500GB of harddisk memory. All the programs were successfuly implemented using C++ language. The g++ compiler was used to compile the C++ programs. Ubuntu 12.04 operating system was used to carry out experimental study. The programs were debugged thoroughly and correct output was obtained. Experimental study was carried out with different character datasets such as Character Trajectories, UJI Pen character, Optical Recognition of Character and MouseTracking character Trajectories datasets.

C. Results and Interpretation

Our proposed PruneMatrix and PruneHash algorithms were tested on different datasets and their execution time and number of nodes generated were recorded. The number of nodes generated using basic distance measures and the proposed pruning techniques were compared and result is

shown in the Table 2. As the length of trajectories is increasing , the number of nodes also increases exponentially. The pruned algorithm reduces the duplicate nodes and number of nodes in the tree are reduced drastically. This lead to the reduction in the execution time of the algorithm.

Trajectory Length	DTW	PruneMatrix
2	19	13
3	94	28
4	481	49
5	2524	76

TABLE 2 : PERFORMANCE OF EDIT DISTANCES NOS OF NODES

Trajectory Length	DTW (ms)	PruneMatrix (ms)
10	0.4	0
11	2.22	0
12	12.38	0
13	71.41	0

TABLE 3: PERFORMANCE OF EDIT DISTANCE EXECUTION TIME

Trajectory Length	PruneMatrix(ms)	PruneMatrix(ms)
100	1.72	0.072
200	28.31	0.28
300	144.46	0.650
400	464.10	1.17

TABLE 4: PERFORMANCE OF PRUNEMATRIX AND PRUNEHASH IN TERM OF EXECUTION TIME.

Execution time of the proposed algorithms were recorded with different length of trajectories. The basic edit distances are taking lot of time and their execution time increases exponentially. The trajectory of length 10 takes 0.40s of execution time whereas length of 13 takes 71.41s. This indicates that, as the length of the time series trajectory increases, its execution time also increases exponentially and it is huge. On the other hand , our proposed prune algorithm takes negligible amount of time for the length of 10,11,12,13 and it is almost zero as shown in the table 3.

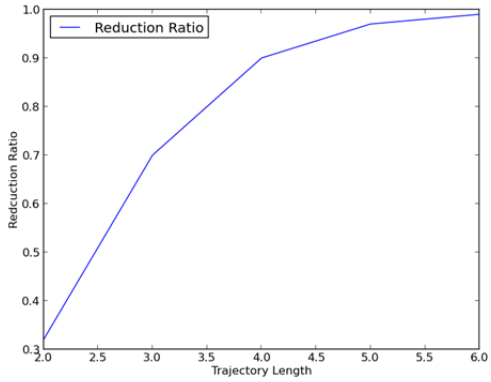

Figure 2: Reduction Ration of Proposed Techniques

We have tested our proposed PruneMatrix and PruneHash algorithms on different set of datasets. PruneMatrix and PruneHash algorithms were tested on time series trajectories of very large length and the results of two algorithms is shown in the table 4. The PruneMatrix algorithm is taking more time compared to PruneHash algorithm. PruneHash is very fast and it is very efficient as compared to PruneMatrix algorithm. Our proposed PruneMatrix and PruneHash algorithms to reduce the duplicate nodes of the tree. The amount of reduction achieved using both the algorithms are very high and same can be seen from the graph shown in Figure 2. As the length of trajectories increases , the reduction ratio is increasing exponentially. Initially the reduction ratio is small but as it crosses trajectory length of 10 , the reduction ratio is almost equal to 99.5 and thereafter it remains constant. The Performance of PruneMatrix and PruneHash algorithms is shown in Figure 3 graphically.

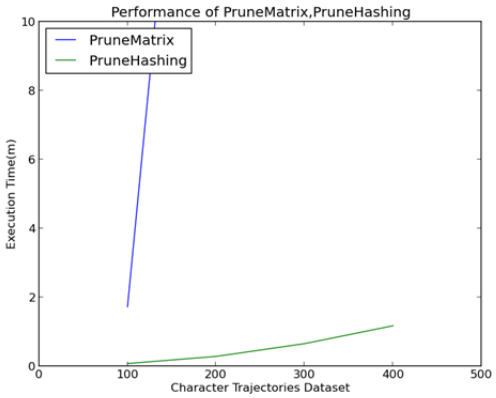

Figure 3: Performance of PruneMatrix and PruneHash

CONCLUSION

We have proposed novel PruneMatrix and PruneHash algorithms to improve the performance of the existing basic distance measures. Our proposed pruning techniques identify duplicate nodes of the tree and pruned the nodes to reduce the time complexity of algorithm. PruneMatrix algorithm is able to eliminate duplicate nodes , thereby improving the performance. PruneHash algorithm avoid searching of node in matrix sequentially and its time complexity is of the order constant O(C) where C is constant whereas PruneMatrix algorithm time complexity is of the order O(N). Our experimental results shows that PruneMatrix and PruneHash algorithms are efficient compared to existing distance measures, further, PruneHash is efficient compared to PruneMatrix algorithm.

REFERENCES

[1] R. Agrawal, C. Faloutsos, A. Swami, Efficient Similarity Search In Sequence Databases, in: FODO, Springer Verlag, 69–84, 1993.

[2] R. Agrawal, K. ip Lin, H. S. Sawhney, K. Shim, Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases, in: In-VLDB, 490–501, 1995.

[3] M. H.W. Kelvin KamWing Chu, Fast Time Series Searching with Scaling and Shifting, in: ACM-PODS, 237–248, 1999.

[4] D. J. Berndt, J. Cli ord, Finding Patterns in Time Series: A Dynamic Programming Approach, in: Advances in Knowledge Discovery and Data Mining, 229–248, 1996.

[5] C. F. Yasushi Sakurai, Masatoshi Yoshikawa, Fast Similarity Search under the Time Warping Distance, in: ACM-PODS, 326–337, 2005.
[6] D. Q. Goldin, T. D. Millstein, A. Kutlu, Bounded similarity querying for time-series data, Information and Computation 194 (2) (2004) 203 – 241, ISSN 0890-5401.
[7] K. Yang, C. Shahabi, An efficient k nearest neighbor search for multivariate time series, Information and Computation 205 (1) (2007) 65 – 98, ISSN 0890-5401.
[8] G. Das, D. Gunopulos, H. Mannila, Time-Series Similarity Problems and Well-Separated Geometric Sets, Nord. J. Comput. 8 (4) (2001) 409–423.
[9] M. Vlachos, G. Kollios, D. Gunopulos, Discovering Similar Multidimensional Trajectories, in: In ICDE, 673–684, 2002.
[10] D. Buzan, S. Sclaro , G. Kollios, Extraction and Clustering of Motion Trajectories in Video, in: 17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004.
[11] M. Vlachos, D. Gunopulos, G. Kollios, Robust Similarity Measures for Mobile Object Trajectories., in: DEXA Workshops, IEEE Computer Society.
[12] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, E. J. Keogh, Indexing MultiDimensional TimeSeries with Support for Multiple Distance Measures, VLDB J. 15 (1) (2006) 1–20
[13] E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrotra, Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases, JOURNAL OF KNOWLEDGE AND INFORMATION SYSTEMS 3 (2000) 263–286.
[14] M. D. Morse, J. M. Patel, An efficient and accurate method for evaluating time series similarity., in: C. Y. Chan, B. C. Ooi, A. Zhou (Eds.), SIGMOD Conference, ACM, ISBN 978-1-59593-686- 8, 569–580.
[15] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast Subsequence Matching in Time-Series Databases, 1994.
[16] B.-K. Yi, H. V. Jagadish, C. Faloutsos, Efficient Retrieval of Similar Time Sequences Under Time Warping, in: S. D. Urban, E. Bertino (Eds.), Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998, IEEE Computer Society, ISBN 0-8186-8289-2, 201–208, 1998.
[17] K.-P. Chan, A. W. chee Fu, Efficient Time Series Matching by Wavelets, in: In ICDE, 126–133, 1999.
[18] B.-K. Yi, C. Faloutsos, Fast Time Sequence Indexing for Arbitrary Lp Norms., in: A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, K.-Y. Whang (Eds.), VLDB, Morgan Kaufmann, ISBN 1-55860-715-3, 385–394,2000.
[19] Y. Cai, R. Ng, Indexing Spatio-temporal Trajectories with Chebyshev Polynomials, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD '04, ACM, New York, NY, USA, ISBN 1-58113-859-8, 599–610, 2004.
[20] E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, M. Cardle, Indexing Large Human-Motion Databases., in: M. A. Nascimento, M. T. zsu, D. Kossmann, R. J. Miller, J. A. Blakeley, K. B. Schiefer (Eds.), VLDB, Morgan Kaufmann, ISBN 0-12-088469-0, 780–791, 2004.
[21] E. Keogh, Exact Indexing of Dynamic Time Warping, 2002.
[22] E. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, M. Cardle, Indexing Large Human-motion Databases, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04, VLDB Endowment, ISBN 0-12-088469-0, 780–791, 2004.
[23] D. Rafiei, A. O. Mendelzon, Querying Time Series Data Based on Similarity, IEEE Transactions on Knowledge and Data Engineering 12 (2000) 675–693.
[24] H. V. Jagadish, Review - Landmarks: a New Model for Similarity-based Pattern Querying in Time Series Databases., ACM SIGMOD Digital Review.
[25] B. Bollobs, G. Das, D. Gunopulos, H. Mannila, Time-Series Similarity Problems and Well-Separated Geometric Sets., in: Symposium on Computational Geometry, 454–456, 1997.
[26] N. Meratnia, R. A. de By, Aggregation and comparison of trajectories., in: A. Voisard, S.-C. Chen (Eds.), ACM-GIS, ACM, ISBN 1-58113-591-2, 49–54, 2002.
[27] B. Lin, J. Su, Shapes based trajectory queries for moving objects., in: C. Shahabi, O. Boucelma (Eds.), GIS, ACM, ISBN 1-59593-146-5, 21–30,2005.
[28] Y. Wu, J. Shen, M. Dai, Traffic object detections and its action analysis., Pattern Recognition Letters 26 (13) (2005) 1963–1984.
[29] J. P. B. Rubio, R. Marfil, A. Bandera, J. A. Rodrguez, L. Molina-Tanco, F. S. Hernndez, Fast gesture recognition based on a twolevel representation., Pattern Recognition Letters 30 (13) (2009) 1181–1189.
[30] M. J. V. Leach, E. P. Sparks, N. M. Robertson, Contextual anomaly detection in crowded surveillance scenes, Pattern Recognition Letters 44 (2014) 71–79.
[31] Z. Chen, H. T. Shen, X. Zhou, Discovering popular routes from trajectories., in: S. Abiteboul, K. Bhm, C. Koch, K.-L. Tan (Eds.), ICDE, IEEE Computer Society, ISBN 978-1-4244-8958-9, 900– 911, 2011.
[32] L.-Y.Wei, Y. Zheng,W.-C. Peng, Constructing popular routes from uncertain trajectories., in: Q. Y. 0001, D. Agarwal, J. Pei (Eds.), KDD, ACM, ISBN 978-1-4503-1462-6, 195–203, 2012.
[33] X. Li, J. Han, J.-G. Lee, H. Gonzalez, Traffic Density-based Discovery of Hot Routes in Road Networks, in: Proceedings of the 10th International Conference on Advances in Spatial and Temporal Databases, SSTD'07, Springer-Verlag, Berlin, Heidelberg, ISBN 978-3-540-73539-7, 441–459, 2007.
[34] M. Lichman, UCI Machine Learning Repository, URL http://archive.ics.uci.edu/ml, 2013.