Correlated Time Series Forecasting using Multi-Task Deep Neural Networks

Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang
Department of Computer Science, Aalborg University, Denmark
{rcirst16, dmicu16, gmures16}@student.aau.dk {cguo, byang}@cs.aau.dk

ABSTRACT

Cyber-physical systems often consist of entities that interact with each other over time. Meanwhile, as part of the continued digitization of industrial processes, various sensor technologies are deployed that enable us to record time-varying attributes (a.k.a., time series) of such entities, thus producing correlated time series. To enable accurate forecasting on such correlated time series, this paper proposes two models that combine convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The first model employs a CNN on each individual time series, combines the convoluted features, and then applies an RNN on top of the convoluted features in the end to enable forecasting. The second model adds additional auto-encoders into the individual CNNs, making the second model a multi-task learning model, which provides accurate and robust forecasting. Experiments on a large real-world correlated time series data set suggest that the proposed two models are effective and outperform baselines in most settings.

KEYWORDS

Correlated time series; Deep learning; Multi-Task Learning

ACM Reference Format:
Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang. 2018. Correlated Time Series Forecasting using Multi-Task Deep Neural Networks. In The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), Oct. 22-26, 2018, Torino, Italy. ACM, NY, NY, USA, 4 pages. https://doi.org/10.1145/3269206.3269310

1 INTRODUCTION

Complex cyber-physical systems (CPSs) often consist of multiple entities that interact with each other. With the continued digitization, various sensor technologies are deployed to record time-varying attributes of such entities, thus producing *correlated time series*.

For example, in an urban sewage system, sensors are deployed to capture time-varying concentration levels of different chemicals (e.g., NO_3 and NH_4) in sewage treatment plants. Different chemicals affect each other due to biological and chemical processes, thus making the different chemical time series correlated. As another example, in a vehicular transportation system, traffic sensors (e.g., loop detectors) are able to capture time-varying traffic information (e.g., in the form of average speeds) of different road segments [6], which produces traffic time series [5]. Since the traffic on a road segment affects the traffic on other road segments, traffic time series on different road segments correlate with each other [2, 13].

Accurate forecasting of correlated time series have the potential to reveal holistic system dynamics of the underlying CPSs, including identifying trends, predicting future behavior [14], and detecting anomalies [9], which are important to enable effective operations of the CPSs. For example, in an sewage system, time series forecasting enables identifying the changing trends of different chemicals, early warning of high concentrations of toxic chemicals, and predicting the effect of incidents, e.g., drought or rains, which enables more effective operations of the sewage system. Similarly, in an intelligent transportation system, analyzing traffic time series enables travel time forecasting, early warning of congestion, and predicting the effect of incidents, which benefit drivers and fleet owners.

To enable accurate and robust correlated time series forecasting, we propose two novel non-linear forecasting algorithms based on deep neural networks—a Convolutional Recurrent Neural Network (CRNN) and an Auto-Encoder Convolutional Recurrent Neural Network (AECRNN). In CRNNs, we first consider each of the correlated time series independently and feed each time series into a 1-dimensional convolutional neural network (CNN). The usage of the CNNs helps us learn features for each individual time series. Next, the convoluted time series features are merged together, which is then fed into a Recurrent Neural Network (RNN), with the aim of learning the sequential information while considering the *correlations* among different time series.

In AECRNNs, we add additional auto-encoders into CRNN. The output of convoluted time series are not only merged together to be fed into the RNN. In addition, each convoluted time series is also reconstructed back to the original time series. Then, the objective function combines the error of the prediction by the RNN and the reconstruction discrepancy by the auto-encoders, making AECRNN a multi-task learning model. The use of auto-encoders makes the CNNs also learn representative features of each time series, but not only distinct features for predicting future values. In other words, the auto-encoders work as an extra regularization which avoids overfitting, disregards outliers, and thus provides more robust forecasting.

To the best of our knowledge, this is the first study that combines CNNs and RNNs in a unified framework with the help of multi-task learning to enable accurate forecasting for correlated time series. Experiments on a large real-word chemical concentration time series from a sewage treatment plant offer evidence that the proposed methods are accurate and robust.

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3269310

2 PRELIMINARIES

A *time series* $X^{(i)} = \langle x_1^{(i)}, x_2^{(i)}, \ldots, x_m^{(i)} \rangle$ is a time-ordered sequence of measurements, where measurement $x_k^{(i)}$ is recorded at time stamp t_k and we have $t_j < t_k$ if $1 \leq j < k \leq m$. Usually, the time interval between two consecutive measurements is constant, i.e., $t_{j+1} - t_j = t_{k+1} - t_k, 1 \leq j, k < m$. A *correlated time series set* is denoted as $X = \langle X^{(1)}, X^{(2)}, \ldots, X^{(n)} \rangle$, where time series in X are correlated with each other. For example, in the sewage treatment example, we have $X = \langle X^{(1)}, X^{(2)}, X^{(3)} \rangle$, where $X^{(1)}$, $X^{(2)}$, and $X^{(3)}$ represent the time series of NH_4, NO_3, and O_2, respectively.

Problem statement: Given a correlated time series set $X = \langle X^{(1)}, X^{(2)}, \ldots, X^{(n)} \rangle$, we aim at predicting the future measurements of a specific, target time series in X. Without loss of generality, the first time series $X^{(1)}$ is chosen as the target time series. More specifically, we assume that the given time series in X have measurements covering a window $[t_{a+1}, t_{a+l}]$ that contains l time stamps, and we aim at predicting time series $X^{(1)}$'s measurements in a future window $[t_{a+l+1}, t_{a+l+p}]$. We call this problem *p-step ahead forecasting*.

3 RELATED WORK

We categorize related studies w.r.t. two dimensions—linear vs. non-linear forecasting and single vs. multiple time series (see Table 1).

	Single	Multiple
Linear	[7]	[11, 14]
Non-Linear	NN	RNN, LSTM, [10, 12], **CRNN, AECRNN**

Table 1: Related Work

We first consider linear methods for single time series. Here, Autoregressive Integrated Moving Average (ARIMA) [7] is widely used as a baseline method. Linear methods for multiple time series also exist, e.g., multiple linear regression [11] and spatio-temporal hidden Markov models [14]. Neural networks (NN) are able to model non-linear relationships, which are often employed to enable non-linear forecasting models. For example, Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) are able to provide non-linear time series forecasting.

In this paper, we aim at providing non-linear forecasting models that are able to support multiple time series, i.e., the *right bottom cell* of Table 1. Multivariate time series convolutional neural network (MTCNN) [12] uses a CNN to extract features from a multivariate time series and then a fully connected neural network layer to predict. We propose CRNN and AECRNN which outperform existing methods in most settings. Similar to MTCNN, we first use CNNs to extract features from multiple time series. However, we instead use RNN in the proposed CRNN and AECRNN. In addition, AECRNN also incorporates auto-encoders, making it a multi-task learning model, which enables effective and robust forecasting. A similar multi-task learning model T2INet [8] employs CNN and auto-encoders to enable classification and clustering, but not for prediction. Non-deep learning methods [10] also exist.

4 PROPOSED MODELS

Convolutional Recurrent Neural Network (CRNN): We propose a CRNN that utilizes a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to enable p-step ahead forecasting for a set of correlated time series. CNNs are used successfully for classifying images by learning the features and patterns of the images. RNNs are able to capture dependencies of a sequence of values, thus being good at forecasting future values. This motivates us to first use CNNs to extract distinctive features of each of the correlated multiple time series, and then to apply RNNs on top of the combined output of CNNs to predict near future values of the target time series. We call this model CRNN (cf. Fig. 1).

Specifically, CRNN takes as input multiple time series, where each input time series contains l measurements. Part A of Fig 1 shows an example where $|X| = 3$ correlated time series $X^{(1)}, X^{(2)}$, and $X^{(3)}$ are fed into CRNN as input. CRNN outputs a single time series, say $Z = \langle z_1, z_2, \ldots, z_p \rangle$, that contains p measurements which are the predicted p measurements of the target time series $X^{(1)}$ in the near future (see Part G of Fig 1).

In the CRNN, we first treat each time series in X independently. In particular, we treat each time series as a $1 \times l$ matrix, as shown in part B of Fig. 1. Next, we apply convolutions on each time series at the convolutional layer (see part C of Fig. 1). In particular, we apply α, e.g., 3, in Fig. 1, filters to conduct convolutions, with the aim to extract distinctive features in the individual input time series. This produces α matrices with size of $1 \times l$. Next, in the pooling layer (see part D of Fig. 1), for each matrix, we apply a max-pooling operator to capture the most representative features of the time series as a $1 \times \frac{l}{2}$ matrix by using a 1×2 window with a stride set to 2. Thus, we obtain a total of α matrices of size $1 \times \frac{l}{2}$ for each input time series. Note that we may apply the convolution and pooling layers multiple times. After convolution and pooling, we have $|X|$ cubes of size $\alpha \times 1 \times \frac{l}{2}$ as the output of the pooling layers (see part D Fig. 1-D). So far, the work of CNNs finishes.

Next, the $|X|$ cubes are concatenated into an n-dimensional vector (see part E of Fig. 1), where $n = |X| \times \alpha \times 1 \times \frac{l}{2}$ to be fed into an RNN (see part F of Fig. 1) for the predicting purpose. We obtain Z as the near future measurements of the target time series (see Part G of Fig 1). The objective function of CRNN is

$$J_1 = \frac{1}{p} \sum_{i=1}^{p} Error(z_i, x_{a+l+i}^{(1)})$$

Here, $Error(\cdot, \cdot)$ is an error function that measures the discrepancy (e.g., mean square error) between the predicted measurement z_i and the ground truth measurement $x_{a+l+i}^{(1)}$ at time stamp $a + l + i$.

Note that the RNN can be easily extended to enable forecasting for all time series, but not just a single target time series.

Auto Encoder CRNN (AECRNN): In AECRNN, we incorporate an auto-encoder to each CNN (see Fig 2). The intuition behind AECRNN is to use auto-encoders to learn robust features and ignore features that represent outliers. The auto-encoders also work as an additional regularization to enable learning most representative features for all input time series but not overfit to features that are specific to forecasting the target time series in the training data.

After the pooling layer, we not only concatenate the output cubes of the pooling layer, but also feed the output cubes to an additional deconvolution layer (see Part E of Fig 2). In particular, each cube is deconvoluted into α matrices with the same size as the matrices in part C. Then, we obtain $|X|$ groups of matrices, where each group has α matrices. Next, we apply Sigmoid activation function to each matrix group to produce a $1 \times l$ matrix. This

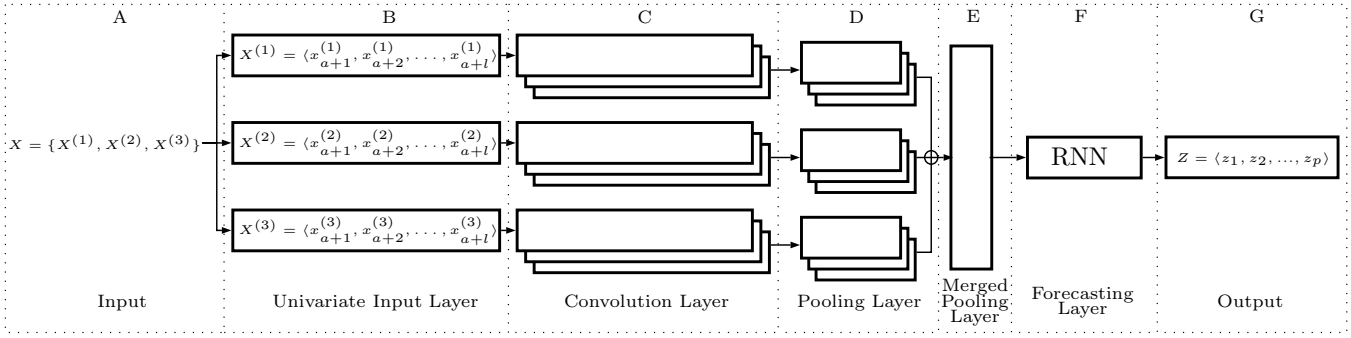

Figure 1: Convolutional Recurrent Neural Network (CRNN)

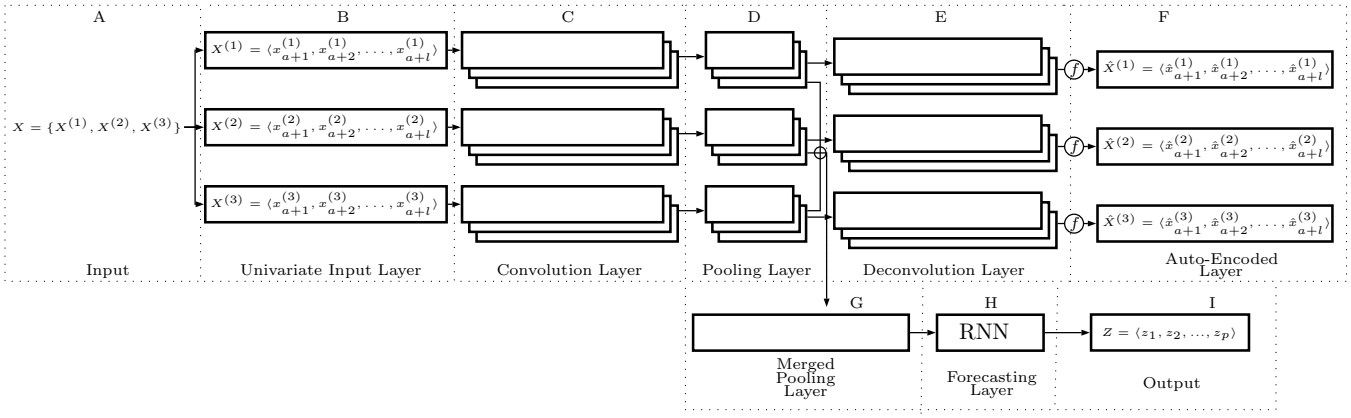

Figure 2: Auto-Encoder Convolutional Recurrent Neural Network (AECRNN)

corresponds $|X|$ reconstructed time series, e.g., $\hat{X}^{(1)}$, $\hat{X}^{(2)}$, and $\hat{X}^{(3)}$, where $\hat{X}^{(i)} = \langle \hat{x}^{(i)}_{a+1}, \hat{x}^{(i)}_{a+2}, \ldots, \hat{x}^{(i)}_{a+l} \rangle$. The objective function of the additional auto encoders is

$$J_2 = \frac{1}{l} \cdot \frac{1}{|X|} \cdot \sum_{k=1}^{|X|} \sum_{i=1}^{l} Error(\hat{x}^{(k)}_{a+i}, x^{(k)}_{a+i}).$$

It measures the discrepancy between the reconstructed measurement $\hat{x}^{(k)}_{a+i}$ and the original, ground-truth measurement $x^{(k)}_{a+i}$ at the $(a+i)$-th time stamp over all $|X|$ time series.

The final objective function of AECRNN is $J = J_1 + J_2$. This makes AECRNN a multi-task learning model, where one task is to forecast the p future measurements of the target time series (i.e., J_1) and the other tasks are to reconstruct $|X|$ time series' l known measurements (i.e., J_2).

5 EMPIRICAL STUDY

Data Sets: We use a large time series data set provided by a sewage treatment center from Aalborg, Denmark. The sewage treatment center has 6 tanks and three different sensors are deployed in each tank to measure the concentrations of three different chemicals, NH_4, NO_3, and O_2, every 2 minutes. The data covers a period of 3 years in total. We consider the three time series of three chemicals from a specific tank as correlated time series.

We choose multiple windows over the 3-year period to test the proposed methods. For each window, we use the first 84% of the data for training and validation, and the remaining 16% of data for testing. When learning, we further segment the training data into multiple training cases using sliding windows. In particular, in each segment, we use a sequence of l measurements as the *input data* to the proposed models and use the immediately following p measurements as the *ground truth data* to calculate the predication errors (i.e., J_1) to enable back-propagation.

Parameters: We vary three *problem parameters* in the experiments according to Table 2, where the default values are shown in bold. Note that the problem parameters define different problem settings and are independent of solutions.

Specifically, we vary the number of time series, i.e., the cardinality of the correlated time series set $|X|$, from 1, 2, to 3. When $|X| = 1$, we only consider a single time series, which is also the target time series. Then, we add 1 and 2 additional correlated time series with the target time series, respectively. Next, we vary the length l of time series when training. We also vary p to enable p-step ahead forecasting, where a large p means forecasting far into the future.

In addition, we vary *solution parameters (a.k.a. hyper-parameters)* that are specific to solutions. In particular, for the proposed deep learning based methods, we vary the number of convolution and pooling layers (1, 2, and 3), number of filters per convolution layer (2, 3, 4, 5, 8, 10, and 16), the filer size (1, 2, 3, 5, and 10), and hidden state size in RNNs (3, 4, 5, 6). We identify the optimal solution parameters for each problem parameter setting.

Baselines: We consider 5 baselines. *Yesterday* is a simple, linear method that propagates the last known value of the time series for the entire prediction window, which is commonly used in financial

$\|X\|$, l, p	YESTERDAY	ARIMA	RNN	LSTM	MTCNN	CRNN	AECRNN
1, 200, 100	16.59±2.70	18.77±2.65	**9.50±1.39**	10.16±2.60	10.49±0.80	10.78±1.26	11.33±2.21
2, 200, 100	16.59±2.70	18.77±2.65	9.43±1.45	10.79±1.61	11.08±0.91	**7.91±0.37**	8.45±0.61
3, 200, 100	16.59±2.70	18.77±2.65	9.59±1.85	10.77±0.57	10.24±1.45	**9.08±1.74**	9.59±0.93
2, 10, 100	18.23±2.93	20.85±3.98	9.86±2.77	**9.12±1.93**	10.01±2.75	9.48±2.44	9.55±2.56
2, 50, 100	16.41±2.25	20.75±5.37	10.77±3.10	11.90±3.15	12.20±3.66	11.83±3.30	**9.56±2.52**
2, 100, 100	17.90±1.58	36.09±24.37	12.62±2.89	12.92±2.39	12.43±1.84	12.78±3.30	**11.04±1.37**
2, 200, 100	16.59±2.70	18.77±2.65	9.43±1.45	10.79±1.61	11.08±0.91	**7.91±0.37**	8.45±0.61
2, 200, 1	1.42±0.25	**0.78±0.05**	5.68±2.04	4.72±1.07	3.54±0.92	3.97±1.40	7.99±5.88
2, 200, 25	12.95±3.35	11.67±2.61	10.04±2.74	9.98±3.20	11.38±1.92	7.42±2.70	**7.12±1.92**
2, 200, 50	17.49±2.47	18.76±2.73	9.42±2.36	9.13±2.45	8.77±0.83	10.06±2.28	**7.81±1.63**
2, 200, 100	16.59±2.70	18.77±2.65	9.43±1.45	10.79±1.61	11.08±0.91	**7.91±0.37**	8.45±0.61

Table 3: Accuracy, MAPE with Standard Deviations

Number of time series $\|X\|$	1, **2**, 3
Training length l	10, 50, 100, **200**
p-step ahead prediction	1, 25, 50, **100**

Table 2: Problem Parameters

time series prediction. Methods *ARIMA*, *RNN*, *LSTM*, and *MTCNN* are covered in Section 3.

Implementation Details: All methods are implemented in Python 3.6, where the deep learning methods are implemented using Tensorflow 1.7. A computer with Intel i7-4700MQ CPU, 4 cores, 16 GB RAM is used to conduct all experiments.

Experimental Results: We use both root mean square error (RMSE) and mean absolute percentage error (MAPE) to evaluate accuracy. Due to space limitation, we report on results only based on MAPE. Results based on RMSE and additional experiments on a different data set are provided elsewhere [1]. Table 3 shows the average MAPE values with standard deviations while varying $|X|$, l, and p. When varying a problem parameter, we keep the other problem parameters with default values according to Table 2.

When $|X| = 1$, i.e., forecasting a single target time series, RNN has the best accuracy. However, when $|X| > 1$, i.e., forecasting the target time series with multiple correlated time series, the proposed CRNN and AECRNN achieve the best accuracy. Note that MTCNN, CRNN, and AECRNN achieve better accuracy when considering correlated times series, meaning that they take advantage of correlated time series. In contrast, Yesterday and ARIMA cannot consider the additional correlated time series, and RNN and LSTM do not show clear improvements. When the input training sequence is very short, e.g., $l = 10$, LSTM gives the best accuracy. As l increases, CRNN and AECRNN achieves better accuracy compared to other methods. This suggests that CRNN and AECRNN are able to take advantages of having longer training sequences. When $p = 1$, ARIMA gives very accurate prediction, which suggests that linear model is very good at short-term forecasting. However, when predicting far into the future , i.e., when p is large, ARIMA deteriorates quickly and CRNN and AECRNN give more accurate forecasting.

Finally, we conducted an experiment to show that AECRNN is able to provide robust forecasting when the input time series (TS) are uncorrelated. In particular, we consider three cases: a single target TS alone, the target TS with a correlated TS, and the target TS with a generated, uncorrelated TS. The results in Table 4 suggest that (1) AECRNN is more robust to deal with the uncorrelated time series, due to the auto-encoders in AECRNN; (2) CRNN is able to take more advantage when having correlated time series.

	CRNN	AECRNN
Single target TS	10.8	10.7
With a correlated TS	8.3	9.1
With an uncorrelated TS	13.0	10.7

Table 4: Dealing with uncorrelated time series, MAPE

6 CONCLUSION AND OUTLOOK

We propose two deep learning models, CRNN and AECRNN, to enable accurate correlated time series forecasting. Experiments on a large real world time series show promising results. In the future, it is of interest to verify the proposed models on time series from different domains such as transportation [3, 4] and to study the scalability of the proposed models for large time series data, e.g., by exploring parallel computing frameworks [15].

REFERENCES

[1] Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang. 2018. Correlated Time Series Forecasting using Deep Neural Networks: A Summary of Results. *CoRR* abs/1808.09794 (2018).

[2] Jian Dai, Bin Yang, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2016. Path Cost Distribution Estimation Using Trajectory Data. *PVLDB* 10, 3 (2016), 85–96.

[3] Zhiming Ding, Bin Yang, Yuanying Chi, and Limin Guo. 2016. Enabling Smart Transportation Systems: A Parallel Spatio-Temporal Database Approach. *TOC* 65, 5 (2016), 1377–1391.

[4] Zhiming Ding, Bin Yang, Ralf Hartmut Güting, and Yaguang Li. 2015. Network-Matched Trajectory-Based Moving-Object Database: Models and Applications. *T-ITS* 16, 4 (2015), 1918–1928.

[5] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *VLDB J.* 27, 2 (2018), 179–200.

[6] Jilin Hu, Bin Yang, Christian S. Jensen, and Yu Ma. 2017. Enabling time-dependent uncertain eco-weights for road networks. *GeoInformatica* 21, 1 (2017), 57–88.

[7] Gareth Janacek. 2010. Time series analysis forecasting and control. *Journal of Time Series Analysis* 31, 4 (2010), 303–303.

[8] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Distinguishing Trajectories from Different Drivers using Incompletely Labeled Trajectories. In *CIKM*. 10 pages.

[9] Tung Kieu, Bin Yang, and Christian S. Jensen. 2018. Outlier Detection for Multi-dimensional Time Series Using Deep Neural Networks. In *MDM*. 125–134.

[10] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2015. The Web As a Jungle: Non-Linear Dynamical Systems for Co-evolving Online Activities. In *WWW*. 721–731.

[11] ML Palomares and D Pauly. 1989. A multiple regression model for prediction the food consumption of marine fish populations. *Marine and Freshwater Research* 40, 3 (1989), 259–273.

[12] Ning Pang, Fengjing Yin, Xiaoyu Zhang, and Xiang Zhao. 2017. A Robust Approach for Multivariate Time Series Forecasting. In *SoICT*. 106–113.

[13] Bin Yang, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2018. PACE: a PAth-CEntric paradigm for stochastic path finding. *VLDB J.* 27, 2 (2018), 153–178.

[14] Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2013. Travel Cost Inference from Sparse, Spatio-Temporally Correlated Time Series Using Markov Models. *PVLDB* 6, 9 (2013), 769–780.

[15] Bin Yang, Qiang Ma, Weining Qian, and Aoying Zhou. 2009. TRUSTER: TRajectory Data Processing on ClUSTERs. In *DASFAA*. 768–771.