SysEvoRecomd: Network Reconstruction by Graph Evolution and Change Learning

Article *in* IEEE Systems Journal · August 2020

DOI: 10.1109/JSYST.2020.2988037

3 authors, including:

Animesh Chaturvedi
Indian Institute of Information Technology, Design and Manufacturing Jabalpur
17 PUBLICATIONS **112** CITATIONS

SEE PROFILE

SysEvoRecomd: Network Reconstruction by Graph Evolution and Change Learning

Animesh Chaturvedi, Aruna Tiwari, and Shubhangi Chaturvedi

***Abstract*— We introduced a *System Evolution Recommender* (SysEvoRecomd) algorithm that uses a novel algorithm *Graph Evolution and Change Learning* (GECL) to do *system network reconstruction*. Internally, GECL uses Deep Evolution Learner (DEL) to learn about evolution and changes happened over a system state series. The DEL is an extension of the deep learning algorithm, which uses an Evolving Connection Matrix (ECM) representing temporal patterns of the *evolving entity-connections* for training incremental states. The DEL generates a *Deep System Neural Network* (Deep SysNN) to do network (graph) reconstruction. The SysEvoRecomd extracts the evolving characteristic of graph with deep neural network techniques. It aims to learn the evolution and changes of the system state series to reconstruct the system network. Our key idea is to design three variants of GECL based on three remodeled deep learning techniques: Restricted Boltzmann Machine (RBM), Deep Belief Network (DBN), and denoising Autoencoder (dA). Based on proposed SysEvoRecomd algorithm, we developed a SysEvoRecomd-Tool, which is applied on different evolving systems: software, natural language, multi-sport event, retail market, and IMDb movie genre. We demonstrated the usefulness of intelligent recommendations using three variants of GECL based on *RBM*, *DBN*, and *dA*.**

***Index Terms*— Machine learning, Graph theory, Systems engineering and theory, and Network reconstruction.**

I. Introduction

AN *evolving system* makes *time variant* (or *non-stationary*) *data* that has many inter-connected entities (or components). Such system evolves to different states over time [1][2][3][4]. A state with inter-connected entities can be represented as a system graph, where each connection is an edge between two entities. The graph can be represented as a matrix, thus each row and column represents connection of an entity. With this, we can perform system evolution analytics on evolving systems [5][6][7].

Usually, different states are built from similar temporal patterns of entity-connections to keep similar functioning of states in an evolving system. The entity-connections relationship can be represented as a directed complex network (graph). In the graph, each entity is a unique vertex (node) and each entity-connection is represented by an edge between the source and target entity. For example, an edge (u, v) means u (source) is connected to v (target); a special case is a loop or cycle (u, u).

Suppose a system state is represented as a graph (first graph of Figure 1) with edges (or connections) {(a, a), (a, b), (a, c), (b, c), (c, a)}. Such that if entity '*a*' is connected to entity '*b*' then intersection between '*a*' (row) and '*b*' (column) has entry '1' else has entry '0'. The first graph is represented as the *first matrix*, such that each row and column represents connection of an entity (as node). This matrix can be converted to its equivalent vector [1, 1, 1, 0, 0, 1, 1, 0, 0]. Let the system evolved to make two more states, which are represented as second and third graphs (represented as second and third matrix).

	a	b	c
a	1	1	1
b	0	0	1
c	1	0	0

	a	b	c
a	1	1	0
b	0	0	1
c	1	0	0

	a	b	c
a	1	1	0
b	0	0	1
c	1	0	1

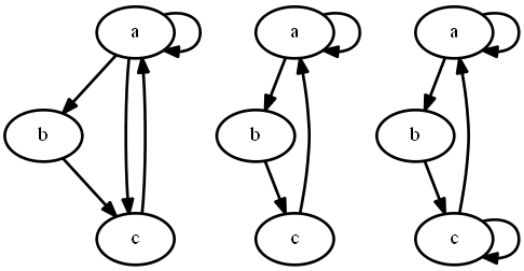

Fig 1. A graph evolution example, when it is evolved in three incremental states with three similar temporal patterns.

Some applications of network reconstruction in the systems are: social networks, bioinformatics, and chemical reaction modelling [8]. Deep Learning techniques [9][10][11] were proven successfully in matrix reconstruction (as an output) e.g. while reconstructing neural circuits (brain-neurons and their mutual contacts). Using reconstruction property of deep learning, we aim to reconstruct matrix of a system network. Our approach can assist in the process of network reconstruction for connections between nodes (entities). This helps to do prediction of system graph structure (e.g. connection prediction) by considering the graph evolution over time.

The motivation behind this work is to introduce an intelligent methodology that learn from the pre-evolved states of a system, and then predict/recommend about the evolving system. Assume any system with many components (or entities) that are connected to each other with a relationship. Due to evolving environment, the system is changed from one state to another. Thus, the entity-connections are also changed; studying such evolution and change would be an interesting problem.

Animesh Chaturvedi is with the Indian Institute of Technology Indore, Indore, MP, India, and with the Department of Informatics, King's College London, Office S2.12 Strand Campus, London, U.K. (e-mail: animesh.chaturvedi88@gmail.com & animesh.chaturvedi@kcl.ac.uk).

Aruna Tiwari is with the Indian Institute of Technology Indore, Indore, MP, India. E-mail: artiwari@iiti.ac.in

Shubhangi Chaturvedi is with the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, Jabalpur, MP, India. E-mail: chaturvedishubhangi51@gmail.com

Section II describes proposed SysEvoRecomd algorithm, which uses our key contributory algorithm Graph Evolution and Change Learning for *system network reconstruction*. Section III describes application of SysEvoRecomd approach on six real-world evolving systems. We discuss related works in Section IV and present conclusions in Section V.

II. System Evolution Recommender

This section describes the proposed *System Evolution Recommender* (SysEvoRecomd), which uses *Graph Evolution and Change Learning* (GECL). Suppose a set of N+1 states has relationship between entities of an evolving system. Let a state series SS = {S_1, S_2, … S_N, S_{N+1}} for various time points T = {t_1, t_2, … t_N, t_{N+1}}, where S_i stands for the i^{th} state in a repository, and *i* varies from 1 to N+1.

The overview of the *Algorithm* ***SysEvoRecomd*** is shown in Figure 2 having abbreviations described in Table I. The SysEvoRecomd preprocesses a set of states to create a $Lists_{N+1}$ with a *mapping* file. Then, it converts the collection of graphs into $Matrices_{N+1}$. After this, $Matrices_N$ are used by GECL algorithm, which produces a *normalized output matrix* M_{NO}. Thereafter, we compare the M_{NO} with *testing matrix* M_T. To facilitate the automation of SysEvoRecomd over broad applications, we developed an intelligent tool, which involves four steps in the following four sub-sections.

Algorithm 1 SysEvoRecomd

Input: *repository*
Retrieve N+1 states (S_1, S_2, … S_N, S_{N+1}) of a *repository*
1: $Lists_{N+1}$ and *mapping* = ***preprocess***(*repository*)
2: $Matrices_{N+1}$ = ***conversion***($Lists_{N+1}$)
3: M_{NO} = ***GECL***($Matrices_N$)
4: *time_series* = ***testing***(M_{NO}, $Matrices_N$, M_T)

A. Preprocessing of a State Series

An evolving system contains entities, which make a graph such that it represents a relationship between entities. Although there are several evolving systems having different pre-processing techniques (depending upon domain), here we describe a general way of preprocessing. Each state is pre-processed to make a graph represented as connection list (i.e. an adjacency list). Each graph contains a set of connections (as edges) between entities (as nodes). The pre-processing algorithm takes N+1 states as input from a directory (retrieved from repository). Process each state (say S_i) to make a connection list (say *connList_i*) in the form *entityName1* connected to *entityName2*. The *connList_i* represents a graph for a state S_i. Store all the N+1 graphs as a connection list in a directory (say $Lists_{N+1}$). Simultaneously, the algorithm also built a mapping file (say *mapping*) that contains two entries: *entityName* and *entityID* for each *entity*. Each entity reappearing in a state series must have the same *entityID* at all the appearances. This mapping file helps to map *entityID* to its *entityName* in post-processing. The pre-processing outputs are N+1 connection lists and one mapping file.

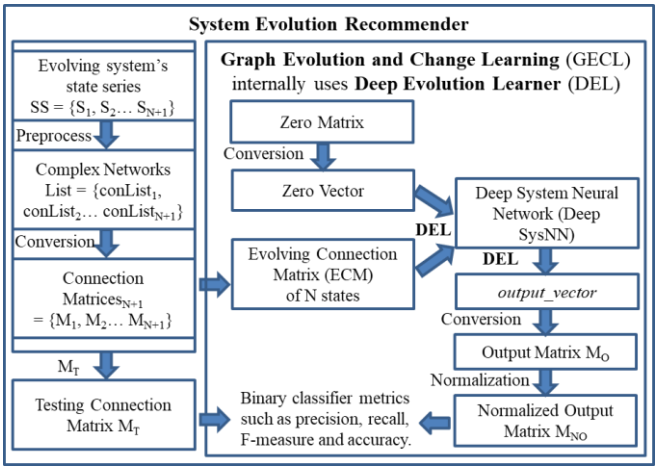

Fig 2. An overview to make a System Evolution Recommender, which uses ECM to generate a Deep System Neural Network.

TABLE I. Information about Abbreviations.

Abbreviation	Its descriptive meaning
$Lists_{N+1}$	collection of N+1 graphs in the form of connection lists
$Matrices_{N+1}$	collection of N+1 connection matrices
$Matrices_N$	collection of N connection matrices
M_O	output matrix
M_{NO}	normalized output matrix
M_T	testing matrix, a matrix not used for training
connList	connection list
conn_matrix	connection matrix

Algorithm 2 preprocess

Input: *repository*
Output: $Lists_{N+1}$ and *mapping*
1: Initialize HashMap *HM*<*entityName*, *entityID*>
2: Initialize integer *counter* = 1
3: Initialize String *buffer* = null
4: **For each** state S_i where *i* ∈ integer 1 to N +1
5: Detect a graph of relationship between entities in the state S_i. The graph is in the form of <*entityName1*, *entityName2*>. Store the graph in a file say *connList_i*
6: **End For**
7: **For each** *connList_i* where *i* ∈ integer 1 to N
8: **Scan** *connList_i*
9: until **end of file** and **Store** it in *buffer*
9: **Scan** *buffer*
10: **If** an *entityName* is in the *HM*
11: In *buffer* replace the *entityName* with its *entityID*
12: **Else**
13: *entityID* = *counter*
14: Add the new tuple <*entityName*, *entityID*> in *HM*
15: In *buffer* replace the *entityName* with its *entityID*
16: Increment counter by 1 i.e. *counter* = *counter* + 1
17: until **End of *buffer***
18: Write *buffer* in *connList_i* and store it in $Lists_{N+1}$
19: **End For**
20: Store the HM in a file named as *mapping*
21: **return** $Lists_{N+1}$ and *mapping*

Algorithm 3 conversion

Input: $Lists_{N+1}$
Output: $Matrices_{N+1}$
1: Initialize integer *counter* = 1
2: Initialize String *buffer* = null
3: **For each** *connList_i* where $i \in$ integer 1 to N
4: Initialize matrix[u][v] = {(0, 0… 0),... (0, 0… 0)}
5: **For each** u ∈ *connList_i* | u is a source node
6: **do while** each v ∈ *connList_i* | v is target node of u
7: *matrix*[u][v] = 1
8: **end do while**
9: **End For**
10: Write *matrix*[u][v] as *conn_matrix_i* in $Matrices_{N+1}$
11: **End For**
12: **return** $Matrices_{N+1}$

B. Conversion

After pre-processing step, each graph (connection list) is converted into its *connection matrix*. The conversion algorithm converts N+1 connection lists ($Lists_{N+1}$ as input) to N+1 connection matrix (stored in a directory (say $Matrices_{N+1}$)). Process each *connList_i* to make a *connection matrices* (say *conn_matrix_i*) and store them in a directory (say $Matrices_{N+1}$), where i represents state such that $1 \leq i \leq N+1$.

C. GECL based on DEL

This subsection presents our key contributory algorithm, *Graph Evolution and Change Learning* (GECL). After encoding a state series as connection matrices, the GECL is used to process a time-variant data $Matrices_N$ (N *connection matrices*) of a state series as input. The GECL algorithm combines multiple *connection matrices* of a state series to form an *Evolving Connection Matrix* (ECM) as an unlabelled data to learn system evolution. Firstly, we provide two formal definitions for the matrices.

Definition 1: A *connection matrix* stores information about connections between entities in the form of a *square binary matrix*. It is a type of *adjacency matrix*, such that presence of connection is represented as '1', whereas '0' represents absence of connection. The size of connection matrix is (m×m), where m is the number of entities. This matrix can be converted to its equivalent row vector, which is referred as *connection vector*. The number of elements in a connection vector is (m×m).

Definition 2: *Evolving Connection Matrix* (ECM) represents an ordered collection of connection vectors for temporally incremental states of a system state series SS = {S_1, S_2, ... S_N}. The ECM represents temporal patterns, which are evolving connections between entities in a system state series. The size of ECM is equal to N×(m×m), where N is the number of states, and (m×m) is the size of a square connection matrix for a state.

Secondly, the GECL algorithm uses the *Deep Evolution Learner* (DEL) that takes ECM as input to learn evolution and changes in the form of temporal patterns. Inside *GECL*, the DEL (as ***deep_evolution_learner***) identifies evolution and changes that happened over states of an evolving system. An ECM is a kind of evolution representor that consists of evolution and change information happened over states. The ECM is useful to learn information as an extended deep learning to make the GECL intelligent. Next, we define DEL.

Definition 3: The *Deep Evolution Learner* (DEL) learns the evolution and changes as temporal patterns of the *evolving entity-connections* in ECM. During training, the ECM can also be visualized as a column matrix of N *connection vectors*, where each connection vector has m × m elements. Internally, the DEL forms a Deep System Neural Network (Deep SysNN), which is defined as follows.

Definition 4: The *Deep SysNN* forms a *weight matrix* to retain information about system evolution based on training (during DEL). The weight matrix of Deep SysNN stores information about evolution and changes as temporal patterns of entity-connections.

Although the deep learning can use any kind of matrix, the DEL uses evolution representor (as ECM), which is the key idea of our approach. The main parameters during training are: learning rate (LR), epoch, and number of hidden units of neurons. After training, the DEL reconstructs a *zero_vector* into an *output_vector* (both the vectors has m×m elements). Thus, the Deep SysNN helps to reconstruct a *zero_vector* (made of a zero matrix M_Z of size m×m) to make an *output_vector*.

Key idea of DEL is to use reconstruction property of deep learning over each connection vector of a state given in ECM (as primary input). This learning helps to reconstruct the entity-connections of a zero vector (as secondary input). The pre-training happens on several connection vectors sequentially containing entity-connections as feature, this initializes *weight matrix* of a *Deep_SysNN* with sensible values. Thereafter, last layer of output units is added on the top of the *Deep_SysNN* and then the whole Deep SysNN is fine-tuned using backpropagation.

Our key contribution is to remodel the existing deep learning techniques. The objective of *deep evolution learner* model is to minimize the connection-vector reconstruction error. The DEL uses existing and successful reconstruction theory of deep learning. We remodeled three well-appreciated deep learning techniques: Restricted Boltzmann Machine (RBM), Deep Belief Networks (DBN), and denoising Autoencoder (dA).

First, the RBM [12][13] is based on the energy model, which can be remodeled for DEL purpose as

$$E(\boldsymbol{ECM}, \boldsymbol{h}) = \sum_i (a_i \, ec_{jk}) + \sum_l (b_l \, h_l) + \sum_i \sum_l (ec_{jk} \, w_{il} \, h_l)$$

where, i represent i^{th} state, $\boldsymbol{ECM}$ represents the matrix of entity-

Algorithm 4 GECL

Input: $Matrices_N$
Output: M_{NO}
1: Initialize a zero matrix as M_Z
2: *zero_vector* = ***matrixToRowVector***(*M_Z*)
3: **For each** *conn_matrix_i* in $Matrices_N$ | $i \in$ 1 to N
4: *conn_vector_i* = ***matrixToRowVector***(*conn_matrix_i*)

$$\text{ECM} = \left\{ \begin{matrix} conn\_vector\_i \\ + \\ ECM \end{matrix} \right\}$$

5: **End for**
6: *output_vector* = ***deep_evolution_learner***(*ECM, zero_vector*)
7: M_O = ***rowVectorToMatrix***(*output_vector*)
8: M_{NO} = ***normalize***(*M_O*)
9: **return** M_{NO}

connections such that ec_{jk} is the entity-connection between entity j and k. The w_{il} is the weight of i and l position in the *weight matrix* of size (m×m) × L for the connections between visible unit $\boldsymbol{ECM}$ (i.e. size of connection vector is m×m) and hidden unit $\boldsymbol{h}$ (user-defined number of hidden layer is L). There are two bias weights a_i for the visible units and b_l for the hidden units. Analogous to the energy model $E(\boldsymbol{ECM}, \boldsymbol{h})$, a probabilistic model can be

$$P(\boldsymbol{ECM}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{ECM},\ \boldsymbol{h})}$$

Thereafter, find conditional probability of h when *ECM* is already given. This makes equation

$$P(\boldsymbol{h}|\boldsymbol{ECM}) = \prod_{l=1}^{L} P(h_l|\boldsymbol{ECM})$$

Now, to identify the best reconstruction, we need to keep least reconstruction error by minimizing the value of

$$-\frac{d log P(\boldsymbol{h}|\boldsymbol{ECM})}{d\boldsymbol{W}_{jkl}} \approx < ec_{jk} h_l >_{reconstruct} - < ec_{jk} h_l >_{data}$$

Second, analogous to the fundamental theory of DBN given by Hinton et al. [10], our model also stack the RBMs together for doing training on ECM. This makes learning in a greedy manner, which forms Deep Belief Network (DBN). Using equation for $P(\boldsymbol{h}|\boldsymbol{ECM})$, we can remodel DBN equation as

$$P(\boldsymbol{ECM}, h^1, h^2 \ldots h^L) = \left(\prod_{l=0}^{L-2} P(h^l|h^{l+1})\right) P(h^{L-1}, h^L)$$

where, L is the total number of hidden layers and h^l is the l^{th} hidden layer.

Third, an auto-encoder [14][15] can be trained to make a deep neural network, which *encodes* the input $\boldsymbol{ECM}$ into some representation $c(\boldsymbol{ECM})$. The $c(\boldsymbol{ECM})$ can be reconstructed by a *decoder* function. Using the given encoding $c(\boldsymbol{ECM})$, we can reformulate minimization of the negative log-likelihood of the connection vector reconstruction as

$$RE = -\log P\big(\boldsymbol{ECM}\big|c(\boldsymbol{ECM})\big)$$

$$RE = -\sum_{i} ec_{jk}\, log f_i\big(c(\boldsymbol{ECM})\big) + \big(1 - ec_{jk}\big) \log\Big(1 - f_i\big(c(\boldsymbol{ECM})\big)\Big)$$

where, $f_i(x)$ is the *decoder* for x_i, and $f_i\big(c(\boldsymbol{ECM})\big)$ is the reconstruction produced by the Deep SysNN for i^{th} state of $\boldsymbol{ECM}$. In order to minimize reconstruction error, we need to minimize the equations of negative log-likelihood.

Vincent et al. [16] introduced denoising Autoencoder (dA) as a stochastic version of the auto-encoder. In the dA, the input (ECM) is stochastically corrupted; however, the uncorrupted ECM is used as target for the reconstruction. For randomly selected subsets of entity-connection temporal patterns, the dA predicts the missing entity-connections from the non-missing entity-connections. Our remodeled training equation for dA is expressed as a reconstruction negative log-likelihood

$$RE = -\log P\left(\boldsymbol{ECM}\middle|c\big(\widetilde{\boldsymbol{ECM}}\big)\right)$$

where $\boldsymbol{ECM}$ is the uncorrupted input, $\widetilde{\boldsymbol{ECM}}$ is the stochastically corrupted input, and c($\widetilde{\boldsymbol{ECM}}$) is encode form of $\widetilde{\boldsymbol{ECM}}$. Again, minimize the negative log-likelihood equation.

After training phase, the DEL minimizes *zero_vector* reconstruction error while making a connection vector named as *output_vector*. In GECL, the *output_vector* is then converted to its equivalent matrix, which is an output matrix M_O. The M_O consists of information about probable existence of a connection between two entities in the ECM. The DEL makes M_O (elements are in between 0 to 1) that further makes *normalized output matrix* M_{NO} (elements are binary). This M_{NO} is the desired output, which represents a reconstructed network.

In short, the GECL algorithm helps to analyze an evolving system. The GECL further depends upon DEL algorithm that uses an ECM made from N evolving states. The DEL learns an ECM to generate a Deep SysNN, which helps to reconstruct a *zero vector* into an *output vector*. The vector reconstruction done by the DEL leads to *system network reconstruction* matrix M_{NO} that might have some inaccuracy. Thus, we need to validate the *system network reconstruction* result (M_{NO}) against an actual system network state (not used for training).

D. Testing

This subsection explains formal evaluation of our recommender system by testing the given recommendations. Testing is done for recommendations provided by M_{NO} about the entity-connections of the evolving systems. The **testing** algorithm measures similarity based on four evaluation metrics: *accuracy*, *precision*, *recall,* and *f-measure*. The algorithm uses the four metrics to compare the M_{NO} with *connection matrices*.

For each state, the values of these four metrics are stored in a file named as *time_series*. For each *i* vary from 1 to N, the algorithm calculates the four metrics values between M_{NO} and *conn_matrix_i*. The state number '*i*' followed by the metrics values are appended as a line in the *time_series* file. Thereafter, the algorithm calculates the four metrics values between M_{NO} and *testing matrix* M_T. The state number 'N+1' of M_T and the four metrics values are appended as a line in the *time_series* file. The *time_series* file aids to produce a (time series) graph, which shows similarity of M_{NO} with the system states and highlights comparison between M_{NO} and M_T. After learning and testing phases of our *SysEvoRecomd*, results are plotted with time series graphs.

Algorithm 5 testing

Input: M_{NO}, $Matrices_N$, M_T
Output: *time_series*
1: Initialize *accuracy, precision, recall, f-measure, time_series*
2: **For each** *conn_matrix_i* in $Matrices_N$ where $i \in$ integer 1 to N
3: *accuracy* = ***accuracy***(M_{NO}, *conn_matrix_i*)
4: *precision* = ***precision***(M_{NO}, *conn_matrix_i*)
5: *recall* = ***recall***(M_{NO}, *conn_matrix_i)*
6: *f-measure* = ***fmeasure***(*precision, recall*)
7: *time_series* = **addLine**(*i, accuracy, precision, f-measure, recall*)
8: **End for**
9: *accuracy* = ***accuracy***(M_{NO}, M_T)
10: *precision* = ***precision***(M_{NO}, M_T)
11: *recall* = ***recall***(M_{NO}, M_T)
12: *f-measure* = ***fmeasure***(*precision, recall*)
13: *time_series* = **addLine**(*N+1, accuracy, precision, f-measure, recall*)
10: **return** *time_series*

III. Experiments on Evolving Systems

This section describes performance evaluation of our SysEvoRecomd-Tool on six evolving systems. For each evolving system, we formed a set of evolving networks, which are further converted to an ECM (evolving connection matrix). Inside GECL algorithm, the DEL algorithm uses the ECM to generate a reconstructed connection matrix (as output), which does *system network reconstruction* based on deep learning approach of matrix reconstruction.

We developed three variants of GECL by using three models of DEL based on: RBM, DBN, and dA (available in Java source code https://github.com/yusugomori/DeepLearning). The M_{NO} represents the reconstructed network, there are three such *system network reconstructions* by three variants of the GECL. To measure the correctness of recommendation by *system network reconstruction*, we compared M_{NO} and M_T (*testing matrix*). For comparison, we used four metrics: *precision*, *recall*, *accuracy*, and *f-measure*. *Mathematically,*

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where, TP is true positives, TN is true negatives, FP is false positives and FN is false negatives. On one hand, the 'true positive' means connection is correctly recommended as connection. The 'true negative' means missing connection is correctly recommended as missing connection. On the other hand, 'false positive' means missing connection is incorrectly recommended as connection. The 'false negative' means connection is incorrectly recommended as missing connection. Both 'true positive' and 'true negative' represents different types of success. Whereas, both 'false positive' and 'false negative' represents different types of failures. F-measure is the harmonic mean of the precision and recall.

$$F - measure = \frac{2 \times precision \times recall}{precision + recall}$$

Next, we discuss experiments on six evolving systems using Figure 3, which helps to study correctness of automatic recommendations done by SysEvoRecomd-Tool. For each (time series), the horizontal axis represents the temporally incremental state series of the evolving system and the vertical axis represents value of metrics between (0-1). The final testing state M_T is unused while training, thus this evidently gives decline performance measure as compared to state used for training. The state series are: version series for HDFS; century series for Bible translation; decade series for Multi-sport events; month series for Frequent market basket; and decade series for Positive and Negative sentiment of IMDB movie genres. The advantages of *system network reconstruction* are derived from time series (in the Figure 3).

Hadoop-HDFS[1]: The size of each *connection matrix* is 3129 × 3129 = 9790641, where 3129 is the number of procedures (entities) in a code version (state) series of HDFS. The GECL algorithm transforms 14 connection matrices into 14 *connection vectors* containing 9790641 elements. The algorithm combines the 14 connection vectors of 14 training states to form an ECM (14 × 9790641) as training input to the DEL. Internally, the DEL generates Deep SysNN that creates an *output_vector* with 9790641 elements. The GECL uses *output_vector* to make a binary matrix M_{NO} of size 3129 × 3129. Then testing algorithm compares the M_{NO} with the testing matrix M_T (version 2.7.2) that contains 2938 number of 1s and 9787703 number of 0s. The reconstructed matrix is similar to all the trained versions (except slightly dissimilarities with some earlier versions: 2.2.0, 2.3.0 etc.). The testing metrics values are slightly less as compared to the last two training versions (2.7.0 and 2.7.1). In the graph, recommendation done by the reconstructed network matrix for testing version ($2.7.2M_T$) is satisfactory.

List of Bible translation[2]: The size of each *connection matrix* is 25 × 25 = 625, where 25 is the number of most frequent words (entities) in a century (state) series. The GECL algorithm transforms 4 connection matrices into 4 *connection vectors* containing 625 elements. The algorithm combines the 4 connection vectors of 4 training states to form an ECM (4 × 625) as training input to the DEL. Internally, the DEL generates Deep SysNN that creates an *output_vector* with 625 elements. The algorithm uses the *output_vector* to make a binary matrix M_{NO} of size 25 × 25. Then testing algorithm compares the M_{NO} with the testing matrix M_T (20 represents years between 2000-2099) that contains 46 number of 1s and 579 number of 0s. The reconstructed matrix is similar to the last two training data of centuries (15-16-17-18 and 19) and dissimilar to the earlier two training data of centuries (7-8-9-10 and 11-12-13-14). The testing metrics values are slightly less as compared to the last two training data of centuries (15-16-17-18 and 19). The recommendation done for the testing century ($20M_T$) by the reconstructed network matrix is satisfactory.

List of multi-sport events[3]: The size of each *connection matrix* is 14 × 14 = 196, where 14 is the number of most frequent words (entities) in a decade (state) series. The GECL algorithm transforms the 7 connection matrices into 7 *connection vectors* containing 196 elements. The algorithm combines the 7 connection vectors of 7 training states to form an ECM (7 × 196) as training input to the DEL. Internally, the DEL generates Deep SysNN that creates an *output_vector* with 196 elements. The algorithm converts the *output_vector* to make a binary matrix M_{NO} of size 14 × 14. Then testing algorithm compares the M_{NO} with the testing matrix M_T (201 i.e. 2010-2017) that contains 8 number of 1s and 188 number of 0s. The reconstructed matrix is 50% (approx.) similar to all the trained decades. The testing metrics values are slightly less than the 50% (approx.) training decade. The recommendation done for the testing decade ($201M_T$) by the reconstructed network matrix is satisfactory for the randomness between temporal patterns in the consecutive states used while training.

Frequent market basket[4]: The size of each *connection matrix* is 118 × 118 = 13924, where 118 is the number of most frequent words (entities) in a month (state) series in the retail market database. The GECL algorithm transforms the 12 connection matrices into 12 *connection vectors* containing 13924 elements. The algorithm combines the 12 connection vectors of 12 training states to form an ECM (12 × 13924) as training input to the DEL. Internally, the DEL generates Deep SysNN that creates an *output_vector* with 13924 elements. The algorithm converts the *output_vector* to make a binary matrix M_{NO} of size 12 × 12. Then testing algorithm compares the M_{NO} with testing matrix M_T (1211 i.e. December 2011) that contains 617 number

1. https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs
2. https://en.wikipedia.org/wiki/List_of_English_Bible_translations
3. https://en.wikipedia.org/wiki/List_of_multi-sport_events
4. https://archive.ics.uci.edu/ml/datasets/Online+Retail

of 1s and 13307 number of 0s. The reconstructed matrix is similar to last three training data of months (911, 1011, and 1111) and dissimilar to the training data of 111 i.e. January month of 2011. The testing metrics values are slightly less as compared to the training data of last eight months. The recommendation done for the testing month ($1211M_T$) by the reconstructed network matrix is satisfactory.

Positive sentiment[5] of movie-name and genres[6] in IMDb: The size of each of the *connection matrix* is 25 × 25 = 625, where 25 is the number of frequent words (entities) in the decade (state) series. The GECL algorithm transforms 11 connection matrices into 11 *connection vectors* containing 625 elements. The algorithm combines 11 connection vectors of 11 training states to form an ECM (11 × 625) as training input to the DEL. Internally, the DEL generates Deep SysNN that creates an *output_vector* with 625 elements. The algorithm uses the *output_vector* to make a binary matrix M_{NO} of size 25 × 25. Then testing algorithm compares the M_{NO} with the testing matrix M_T (201 i.e. 2010-2019) that contains 47 number of 1s and 578 number of 0s. The reconstructed matrix is similar to last two training decades (1990s and 2000s) and slightly dissimilar with training decades (1910s, 1920s, 1940s, and 1950s). The testing metrics values are slightly less as compared to the last training decade 200 i.e. 2000-2009. Recommendation done for the testing decade ($201M_T$) by the reconstructed network matrix is satisfactory even for the randomness between temporal patterns in the consecutive states while training.

Negative sentiment[5] of movie-name and genres[6] in IMDb: The size of each of the *connection matrix* is 45 × 45 = 2025, where 45 is the number of frequent words (entities) in the decade (state) series. The GECL algorithm transforms 11 connection matrix into 11 *connection vectors* containing 2025 elements. The algorithm combines 11 connection vectors of 11 training states to form an ECM (11 × 2025) as training input to the DEL. Internally, the DEL generates Deep SysNN that creates an *output_vector* with 2025 elements. The algorithm uses the *output_vector* to make a binary matrix M_{NO} of size 45 × 45. Then testing algorithm compares the M_{NO} with the testing matrix M_T (201 i.e. 2010-2019) that contains 177 number of 1s and 1848 number of 0s. The reconstructed matrix is similar to last two trained decades (1990s and 2000s) and dissimilar with earlier decades (1900s, 1910s, 1920s, 1930s, 1940s, 1950s, and 1960s). The similarity of the reconstructed matrix is progressive with respect to the states (time points). The testing metrics values are slightly less as compared to the last trained decade 200 i.e. 2000-2009. Recommendation done for the testing decade ($201M_T$) by the reconstructed network matrix is satisfactory even for the randomness between temporal patterns in the consecutive states of the training dataset.

IV. Related Works and Discussions

This section presents state-of-the-art works. Previously, Chaturvedi et al. [17] presented an abstract on system evolution recommendation. In graph learning, Nori *et al.* [18] proposed ActionGraph, which is used to predict interests of social media users based on time-evolving and multinomial relational data. Li *et al.* [19] proposed a RBM model for representation learning on linked data. Yang *et al.* [20] presented a framework, Concept Graph Learning (CGL), for the academic graphs of courses and concepts using course links onto the concept links. Yanardag, and Vishwanathan [21] presented Deep Graph Kernels, a framework that learns latent representations of sub-structures for graphs by learning dependency information between sub-structures. Cao and Xu [22] proposed a model for graph learning based on stacked denosing autoencoders by encoding each vertex as a low dimensional vector. Bui *et al.* [23] proposed Neural Graph Machines with a regularized graph that applies neural networks on label/unlabeled data with Feed-forward NNs, CNNs, and LSTM RNNs. Somanchi et al. [24] presented an approach of Learning Graph Structure (LGS), which is a greedy framework for efficient learning of graph structure. In contrast, we presented deep learning of evolving system over time using fundamental network (graph) theory.

Recently, Angulo *et al.* [25] found that reconstructing any property of the interaction matrix is generally difficult, and presented fundamental limitations to design algorithms for better network reconstruction. Most recently, Porfiri and Marin [26] used finite-state ergodic Markov chain model and described an approach for entropy-based network reconstruction. Whereas, we exploited the property of deep learning to reconstruct interaction (connection) matrix for achieving *system network reconstruction*.

Applications of our approach include network reconstruction for connections between nodes (entities). Tano *et al.* [27] described a method for dynamic network reconstruction by collecting node processing information to maintain an efficient network for changing system. Yue *et al.* [28] addressed and solved the problem of reconstruction of linear dynamic networks from heterogeneous datasets using regression model. The SysEvoRecomd can improve thinking about system evolution, thus leads to improve the system thinking (Whitehead *et al.* [29]). Earlier the systems thinking is extended with compressed sensing to the sparse network reconstruction [30]. Our approach is applicable to different system evolution studies, e.g. nervous system evolution [31], file system evolution [32], and solar system evolution [33].

V. Conclusion

This paper introduced a *System Evolution Recommender* (SysEvoRecomd) algorithm that internally uses *Graph Evolution and Change Learning* (GECL). The GECL algorithm processes a series of graphs using *Deep Evolution Learner* algorithm, which uses an *Evolving Connection Matrix* and outputs a *Deep System Neural Network*. We developed an automated tool SysEvoRecomd-Tool, which did promising *system network reconstruction*. Our approach is end-to-end implementable on Java like real time computing environment, and applicable for natural language and software systems. For example, our approach can be applied to systems based on multimedia recommendation and generation [34][35][36]. Our SysEvoRecomd is novel, useful, and sensible as it uses inherent matrix reconstruction characteristics of remodeled deep learning. It has applications in system evolution analysis, system network reconstruction, and recommender system development. In the future, it is possible to do statistical stochastic approximation of uncertainty in network completion and propagation of this uncertainty to the next state.

5. https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
6. http://www.imdb.com/interfaces/

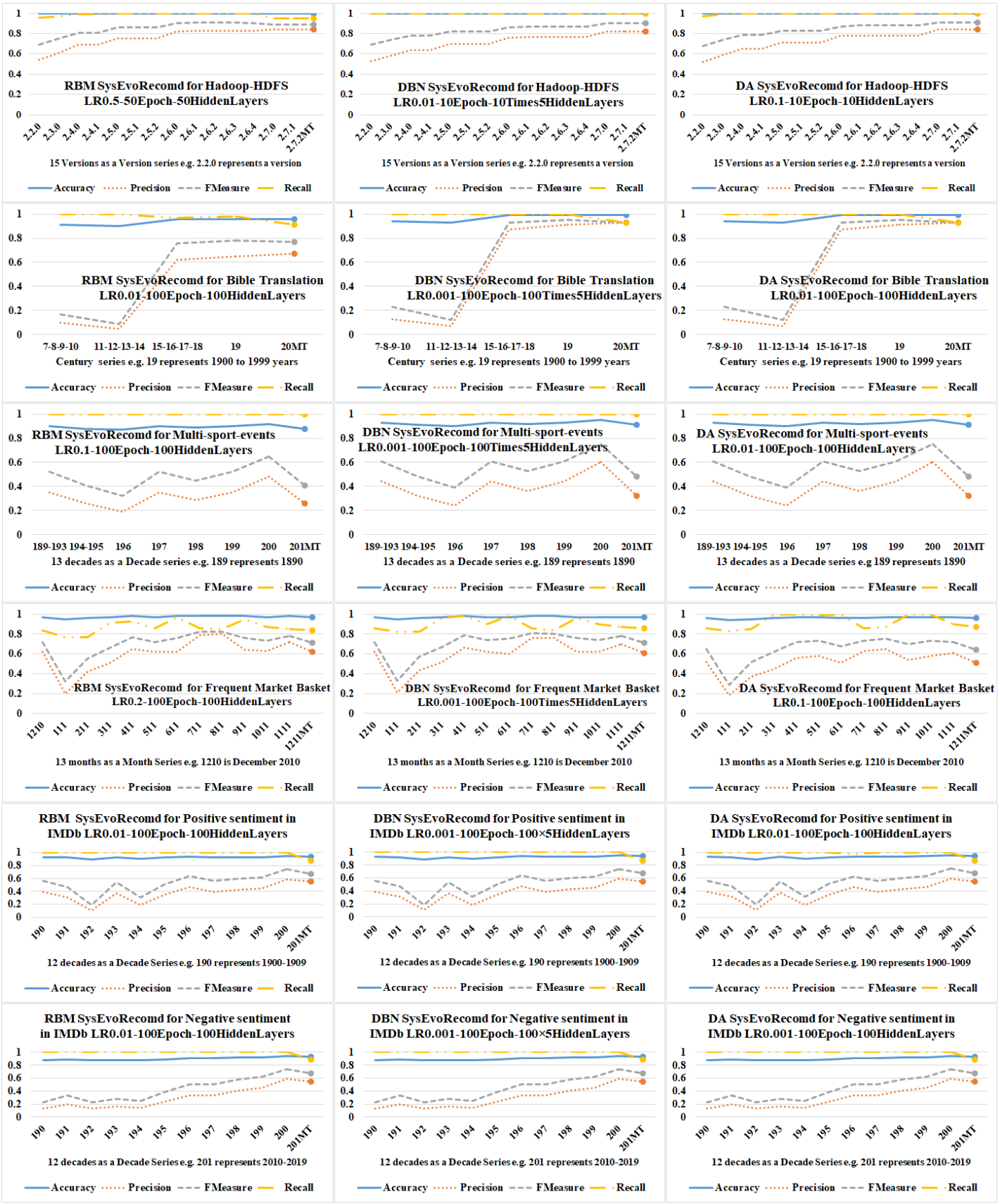

Fig. 3. The figure shows results for experiments on six evolving systems. Each figure shows three (time series) graphs (in each row) for three variants of GECL applied on an evolving system. Each graph contains four time series based on four binary classifier metrics: accuracy, precision, f-measure, and recall. Each data point (coordinate (x, y)) of a time series denotes a metric value for similarity between normalized output matrix (M_{NO}) and connection matrix of temporally incremental states (in horizontal axis). Last coordinate of each time series denotes a metric value for similarity between M_{NO} and testing matrix (M_T). This helps to study correctness of automatic recommendations done by SysEvoRecomd-Tool.

ACKNOWLEDGEMENT

The authors wish to thank Dr. Yi Yu (National Institute of Informatics (NII), Tokyo, Japan) for her advice on the presentation of the paper.

REFERENCES

[1] B. Heydari, and K. Dalili. "Emergence of modularity in system of systems: Complex networks in heterogeneous environments." *IEEE Systems Journal* 9.1 (2015): 223-231.

[2] A. M. Ross, D. H. Rhodes, and D. E. Hastings. "Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value." *Systems Engineering* 11.3 (2008): 246-262.

[3] S. A. Frost, and M. J. Balas. "Evolving Systems and Adaptive Key Component Control". *In: Arif, T.T. (ed.) Aerospace Technologies Advancements.* ISBN: 978-953-7619-96-1 (2010).

[4] P. Angelov, and N. Kasabov. "Evolving intelligent systems, eIS." *IEEE SMC eNewsLetter* 15 (2006): 1-13.

[5] A. Chaturvedi and A. Tiwari. "System Evolution Analytics: Deep Evolution and Change Learning of Inter-Connected Entities". *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)* 2018: 3075-3080.

[6] A. Chaturvedi and A. Tiwari. "System Evolution Analytics: Evolution and Change Pattern Mining of Inter-Connected Entities". *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)* 2018: 3877-3882.

[7] A. Chaturvedi and A. Tiwari. "System Network Complexity: Network Evolution Subgraphs of System State series". *IEEE Transactions on Emerging Topics in Computational Intelligence*.

[8] K. P. Unnikrishnan, et al. "Network reconstruction from dynamic data." *ACM SIGKDD Explorations Newsletter* 8.2 (2006): 90-91.

[9] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

[10] G. E. Hinton et al. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[11] Y. Bengio, et al. "Greedy layer-wise training of deep networks." *Advances in Neural Information Processing Systems* 19 (2007): 153.

[12] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. No. CU-CS-321-86. Colorado Univ at Boulder Dept of Computer Science, 1986.

[13] G. W. Taylor et al. "Modeling human motion using binary latent variables." *Advances in Neural Information Processing Systems*. 2006.

[14] Y. Bengio, P. Lamblin, D. Popovici, & H. Larochelle. "Greedy layer-wise training of deep networks." *Advances in Neural Information Processing Systems 19 (NIPS'06)*, MIT Press, (2007): 153–160.

[15] M. Ranzato, C. Poultney, S. Chopra, & Y. LeCun. "Efficient learning of sparse representations with an energy-based model." *Advances in Neural Information Processing Systems 19 (NIPS'06)*, MIT Press, (2007): 1137–1144.

[16] P. Vincent, et al. "Extracting and composing robust features with denoising autoencoders." *25th Int. Conf. on Machine learning*. ACM, 2008.

[17] A. Chaturvedi and A. Tiwari. "SysEvoRecomd: Graph Evolution and Change Learning based System Evolution Recommender". *IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2018: 1499-1500.

[18] N. Nori, D. Bollegala, and M. Ishizuka. "Interest prediction on multinomial, time-evolving social graph." *IJCAI*. Vol. 11. 2011.

[19] K. Li, et al. "LRBM: A restricted boltzmann machine based approach for representation learning on linked data." *2014 IEEE Int. Conf. on. Data Mining (ICDM)*, IEEE, 2014.

[20] Y. Yang, et al. "Concept graph learning from educational data." *8th ACM Int. Conf. on Web Search and Data Mining*. ACM, 2015.

[21] P. Yanardag, and S. V. N. Vishwanathan. "Deep graph kernels." *21th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*. ACM, 2015.

[22] S. Cao, W. Lu, and Q. Xu. "Deep Neural Networks for Learning Graph Representations." *AAAI*. 2016.

[23] T. D. Bui, S. Ravi, and V. Ramavajjala. "Neural Graph Machines: Learning Neural Networks Using Graphs." *arXiv preprint arXiv:1703.04818* (2017).

[24] S. Somanchi, and D. B. Neill. "Graph Structure Learning from Unlabeled Data for Early Outbreak Detection." *IEEE Intelligent Systems* 32.2 (2017): 80-84.

[25] M. T. Angulo, et al. "Fundamental limitations of network reconstruction from temporal data". *Journal of The Royal Society Interface* 14.127 (2017): 20160966.

[26] M. Porfiri, and M. R. Marin. "Information flow in a model of policy diffusion: an analytical study." *IEEE Trans. on Network Science and Engineering* (2017).

[27] S. Tano, et al. "Dynamic reconstruction method for discrimination network." U.S. Patent No. 4,761,746. 2 Aug. 1988.

[28] Z. Yue, et al. "Linear Dynamic Network Reconstruction from Heterogeneous Datasets." *IFAC-PapersOnLine* 50.1 (2017): 10586-10591.

[29] Whitehead N. Peter, William T. Scherer, and Michael C. Smith. "Systems thinking about systems thinking a proposal for a common language." *IEEE Systems Journal* 9.4 (2015): 1117-1128.

[30] A. M. Madni, "A systems perspective on compressed sensing and its use in reconstructing sparse networks." *IEEE Systems Journal* 8.1 (2013): 23-27.

[31] J. E. Niven, and L. Chittka. "Evolving understanding of nervous system evolution." (2016): R937-R941.

[32] L. Lu, et al. "A study of Linux file system evolution." *ACM Transactions on Storage (TOS)* 10.1 (2014): 3.

[33] S. R. Taylor. *Solar system evolution: A new perspective*. Cambridge University Press, 2001.

[34] Y. Yu, et al. "Category-based deep CCA for fine-grained venue discovery from multimodal data." *IEEE Transactions on Neural Networks and Learning Systems* 30.4 (2018): 1250-1258.

[35] Y. Yu, et al. "Deep cross-modal correlation learning for audio and lyrics in music retrieval." *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 15.1 (2019): 1-16.

[36] Y. Yu,, and Simon Canales. "Conditional LSTM-GAN for Melody Generation from Lyrics." *arXiv preprint arXiv:1908.05551* (2019).

Animesh Chaturvedi is post-doctoral research-assistant in NetSys Group of King's College London, and working with The Alan Turing Institute. He had submitted thesis towards the PhD degree at the IIT Indore. He received the BEng degree from IET - Devi Ahilya University, and the MTech degree from Indian Institute of Information Technology, Design and Manufacturing, Jabalpur. He did research work with IIT-Kanpur, Motorola, and Arris. He has been reviewer for IEEE TETC, IEEE TBD, and IEEE SJ. His research interest is in Data mining, Machine Learning, and Evolving systems. He attended events including 29th IEEE ICSM 2013, IEEE CloudCom 2014, 36th IEEE/ACM ICSE 2014, and Heidelberg Laurate Forum (HLF) 2019.

Aruna Tiwari is an Associate Professor of Computer Sci. and Eng. at IIT Indore, since 2012. She received BEng, MEng, and PhD degrees in Computer Eng. Her research interests include neural network, fuzzy clustering, and evolutionary computation. She has been reviewer for many journals & conferences. She has research collaboration with CSIR CEERI Pilani and Indian Institute of Soyabean Research (Indian Council of Agriculture & Research).

Shubhangi Chaturvedi is working towards the PhD degree at the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur. She received the BEng degree from RGPV University, and the MTech degree from National Institute of Technology Bhopal, Bhopal. She has been Assistant Professor of National Institute of Technology Bhopal, Bhopal. Her research interest is in Data mining and Big Data Analytics.